

AD-A206 664

RADC-TR-88-167
Final Technical Report
July 1988



ALGEBRAIC INTEGER QUANTIZATION AND CONVERSION

The MITRE Corporation

Richard A. Games, Sean D. O'Neill and Joseph J. Rushanan

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

DTIC
SELECTED
APR 10 1989
S D
D C8

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

89 4 10 021

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-167 has been reviewed and is approved for publication.

APPROVED:

Arnold E. Argenzia
ARNOLD E. ARGENZIA
Project Engineer

APPROVED:

John D. Kelly
JOHN D. KELLY
Acting Technical Director
Directorate of Communications

FOR THE COMMANDER:

James W. Hyde III
JAMES W. HYDE III
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DCCR) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS N/A		
2a. SECURITY CLASSIFICATION AUTHORITY N/A		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MTR - 10337		5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-167		
6a. NAME OF PERFORMING ORGANIZATION The MITRE Corporation		6b. OFFICE SYMBOL (if applicable) DCCR	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (DCCR)	
6c. ADDRESS (City, State, and ZIP Code) Dept D82 Burlington Road Bedford MA 01730		7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center		8b. OFFICE SYMBOL (if applicable) DCCR	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-86-C-0001	
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		10. SOURCE OF FUNDING NUMBERS		
		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
		000000	MOIE	76
				60
11. TITLE (Include Security Classification) ALGEBRAIC INTEGER QUANTIZATION AND CONVERSION				
12. PERSONAL AUTHOR(S) Richard A. Games, Sean D. O'Neil, and Joseph J. Rushanan				
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM Oct 86 TO Oct 87		14. DATE OF REPORT (Year, Month, Day) July 1988	15. PAGE COUNT 142
16. SUPPLEMENTARY NOTATION N/A				
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Residue Number System Algebraic - Integer Number System Signal Processing		
FIELD 12	GROUP 01			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The algebraic-integer number representation, in which the signal sample is represented by a set of (typically four to eight) small integers, combines with residue number system (RNS) processing to produce processors composed of simple parallel channels. The analog samples must first be converted into the algebraic-integer representation, and the final algebraic-integer result converted back to an analog or digital form. These are quantization problems. Second, the algebraic-integer representation must be converted into and out of two levels of RNS parallelism. These are RNS conversion problems. This paper provides practical solutions, which can be implemented with current technology, to these quantization and conversion problems. K. Games, S. D. O'Neil, J. J. Rushanan (KR)				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Arnold E. Argenzia		22b. TELEPHONE (Include Area Code) (315) 330-3091	22c. OFFICE SYMBOL RADC (DCCR)	

ACKNOWLEDGMENTS

The authors would like to thank Carol L. Nowacki, who designed the mixed-radix converter referred to in section 4.3.1, and John J. Vaccaro for their help in completing the area comparisons of section 4.3. Steve J. Meehan is responsible for the fractional-representation converter design referred to in section 4.3.2. Also, thanks to Mike G. Butler and John J. Sawyer for their contributions to the VLSI aspects of the work in sections 4 and 5, and to Daniel Moulin for keeping the hardware concerns honest. The design of the $A + BC$ cell of section 5.4.2 follows that of the chip called MULTISCALE, a programmable modulo multiplier chip, designed by Gary Doodlesack. Stephen C. Atkins, Curtis P. Brown, and Stephen C. Lee provided programming support for the project. Also, thanks to Dean O. Carhoun and Edwin L. Key for their continued interest and support of this work. Finally, the authors wish to acknowledge both Michael F. Bridgland and Michael D. Houle for their help in typesetting this document.

Accesion For	
NTIS	CRA&I <input checked="" type="checkbox"/>
DTIC	TAB <input type="checkbox"/>
Unannounced <input type="checkbox"/>	
Justification	
By _____	
Distribution / _____	
Availability Codes	
Dist	Avail and / or Special
A-1	

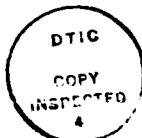


TABLE OF CONTENTS

SECTION	PAGE
1. Introduction	1
2. The Algebraic-Integer Number Representation and Residue Number System Processing	5
2.1 Algebraic-Integer Extensions	5
2.1.1 Adjoining $\sqrt{2}$	5
2.1.2 Adjoining $e^{2\pi i/8}$	7
2.1.3 Adjoining $\sqrt{2 + \sqrt{2}}$	11
2.1.4 Adjoining $e^{2\pi i/16}$	12
2.2 Residue Number System Processing with Algebraic Integers	14
2.2.1 Integer RNS Processing	14
2.2.2 Algebraic-Integer RNS Processing	15
2.3 History of Algebraic-Integer Processing	16
2.4 Conclusion	17
3. Analog-to-Algebraic-Integer Conversion	19
3.1 Direct Analog-to-Algebraic-Integer Conversion	19
3.1.1 Compressor Characteristic Method	19
3.1.2 Successive Approximation Method	21
3.2 Two-Stage Analog-to-Algebraic-Integer Conversion	25
3.3 Complex Algebraic-Integer Quantization	27
3.4 Quantization Performance for $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$	28
3.4.1 Uniform Inputs	29
3.4.2 Gaussian Inputs	33
3.5 Conclusion	35
4. Integer RNS Conversion: The Outer Level of Parallelism	39
4.1 Full-Precision Output Conversion	40
4.1.1 Fractional Representation	41
4.1.2 Modified Fractional Representation	43
4.2 Adjustable Precision Output Conversion	45
4.2.1 Fractional Representation	45
4.2.2 Modified Fractional Representation	47
4.3 Comparison of Conversion Methods	48
4.3.1 Mixed Radix vs. Fractional Representation	49
4.3.2 Advantages of Modified Fractional Representation	51

SECTION	PAGE
4.4 Conclusion	52
5. Polynomial RNS Conversion: The Inner Level of Parallelism	53
5.1 Conversion as Matrix Multiplication	53
5.1.1 Description of the Conversion	53
5.1.2 LU Decomposition of the Vandermonde Matrix	55
5.1.3 An Example: $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$, $p = 31$	57
5.1.4 Change of Basis	58
5.2 The LU Architecture	59
5.2.1 Data Flow	60
5.2.2 The $A + BC$ Cell	64
5.2.3 The $C(A - B)$ Cell	71
5.3 Systolic Conversion	72
5.3.1 The Systolic Architecture	73
5.3.2 Comparison with LU Architecture	76
5.4 VLSI Implementation	76
5.4.1 Chip Layout Limitation	77
5.4.2 Algorithms for the $A + BC$ Cell	77
5.4.3 Layout and Testing	81
5.5 Conclusion	81
6. Algebraic-Integer-to-Analog (or Digital) Conversion	87
6.1 Algebraic-Integer Requantization Performance	88
6.2 An Error Analysis	92
6.3 Complex Algebraic-Integer Requantization	95
6.4 Implementation Issues	96
6.5 Conclusion	97
7. Future Work and Conclusion	99
7.1 Algebraic-Integer Brassboard	99
7.2 Performance of an Algebraic-Integer FIR Filter	100
7.3 Conclusion	105
Appendix A LU Decomposition	109
Appendix B Some Algebraic Integers	113
List of References	127

LIST OF ILLUSTRATIONS

FIGURE	PAGE
1. RNS Processing with Algebraic Integers	2
2. $\mathbb{Z}[\sqrt{2}]_4$	7
3. Basis Vectors for $\mathbb{Z}[\omega]$	8
4. $\mathbb{Z}[e^{2\pi i/8}]_2$	10
5. $\mathbb{Z}[e^{2\pi i/16}]_2$	13
6. Compressor Characteristics for Real Algebraic Integers	20
7. Analog-to- $\mathbb{Z}[\sqrt{2}]_{\{-2,-1,0,1\}}$ Converter	23
8. $\mathbb{Z}[\sqrt{2}]_{\{-2,-1,0,1\}}$ Converter Functions	26
9. Uniform Inputs; $\mathbb{Z}[\sqrt{2+\sqrt{2}}]_M$, $2 \leq M \leq 12$; Nonpolynomial Basis; Direct Quantizer	30
10. Uniform Inputs; $\mathbb{Z}[\sqrt{2+\sqrt{2}}]_M$, $2 \leq M \leq 12$; Nonpolynomial Basis; Two-Stage Quantizer	31
11. Gaussian Inputs; $\mathbb{Z}[\sqrt{2+\sqrt{2}}]_M$, $2 \leq M \leq 12$; Nonpolynomial Basis; Direct Quantizer	35
12. Gaussian Inputs; $\mathbb{Z}[\sqrt{2+\sqrt{2}}]_M$, $2 \leq M \leq 12$; Nonpolynomial Basis; Two-Stage Quantizer	36
13. Residue Decoding by the Chinese Remainder Theorem	40
14. Four-Modulus Mixed-Radix Converter	50
15. Four-Modulus Fractional-Representation Converter	51
16. Flow of Variables in LU Grid ($n = 4$)	62
17. Flow of Variables in UL Grid ($n = 4$)	63
18. The $A + BC$ Cell	64
19. Grid for LU Computation with $A + BC$ Cell	66
20. A Specific Example of LU Computation	67
21. Grid for UL Computation with $A + BC$ Cell	69
22. A Specific Example of UL Computation	70
23. The $C(A - B)$ Cell	71
24. Data Flow in the Systolic Architecture	74
25. The Systolic Architecture with $A + BC$ Cells	75
26. A Modulo Reduction Example (16-bit Adders)	79
27. A Modulo Reduction Example (8-bit Adders)	80
28. The Multiplier/Accumulator	83

FIGURE	PAGE
29. The Modulo Reduction	84
30. The $A + BC$ Cell	85
31. The LU Architecture Chip	86

LIST OF TABLES

TABLE	PAGE
1. Elements of $\mathbb{Z}[\sqrt{2}]$ with Decreasing Magnitude	6
2. Binary Codes for $\mathbb{Z}[\sqrt{2}]_{\{-2,-1,0,1\}}$	22
3. Comparison of Direct and Two-Stage $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]_M$ Quantizers, $2 \leq M \leq 12$; Nonpolynomial Basis; Uniform Inputs	31
4. Efficiency of $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$; Uniform Inputs; Direct Quantizer	32
5. Efficiency of $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$; Uniform Inputs; Direct Quantizer; Variance Mismatch of $1/4$	33
6. SNR of Optimal Uniform Quantizer; Gaussian Input	34
7. Comparison of Direct and Two-Stage $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]_M$ Quantizers, $2 \leq M \leq 12$; Nonpolynomial Basis; Gaussian Inputs	36
8. Efficiency of $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$; Gaussian Inputs; Direct Quantizer	37
9. Optimal Approximations for b -bit Requantization for $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$	90
10. SNR Degradation for b -bit Polynomial Evaluation	91
11. Range vs. SNR for Algebraic Integers	103
12. Range vs. SNR for Integers	103
13. Summary of Largest Ranges	104

SECTION 1

INTRODUCTION

A requirement of digital signal processing is that analog (voltage) signals are sampled and then converted to sequences of numbers. In the conventional quantization methods, a given signal-sample is converted (by scaling and rounding) to a single integer whose value is proportional to the signal's voltage; the subsequent processing manipulates these integers, and produces a final result (another integer) that again is proportional to the corresponding analog output.

Modern algebra and number theory offer many alternatives to this simple way of digitally representing numbers, and some alternatives lead to processors that are simpler, faster, and easier to design and test. In the residue number system (RNS) approach, for example, each integer is replaced by a set of smaller integers, and then processing (consisting of additions and multiplications) is performed by manipulating these smaller integers independently in parallel.

Processing with *algebraic integers*, in which the signal sample is represented by a set of (typically four to eight) small integers, was introduced in [3]. This approach, when combined with RNS processing, leads naturally to processors composed of simple parallel processing channels, and is especially well suited to situations involving information with both amplitude and phase. Roughly speaking, algebraic integers can be used to add a second level of parallelism to integer RNS processing. RNS processing with algebraic integers is depicted schematically in figure 1.

Two types of problems must be resolved before the method can be used in practice. First, the analog samples must be converted into the algebraic-integer representation, and the final algebraic-integer result converted back to an analog output. These are quantization problems. Second, the algebraic-integer representation must be converted into and out of the two levels of RNS parallelism, corresponding to the outer and inner channels in figure 1. These are RNS conversion problems.

This paper addresses these quantization and conversion problems and proposes solutions that can be implemented with current technology. The solutions can be programmed for various choices of algebraic integers and RNSs. Once the quantization and conversion structures/designs that are contained in this paper are implemented, subsequent algebraic-integer RNS processing can be completed by merely developing the RNS processing channels. These channels are the same as in integer RNSs, and the possibility exists, with some restrictions, of utilizing processing channels that were developed previously.

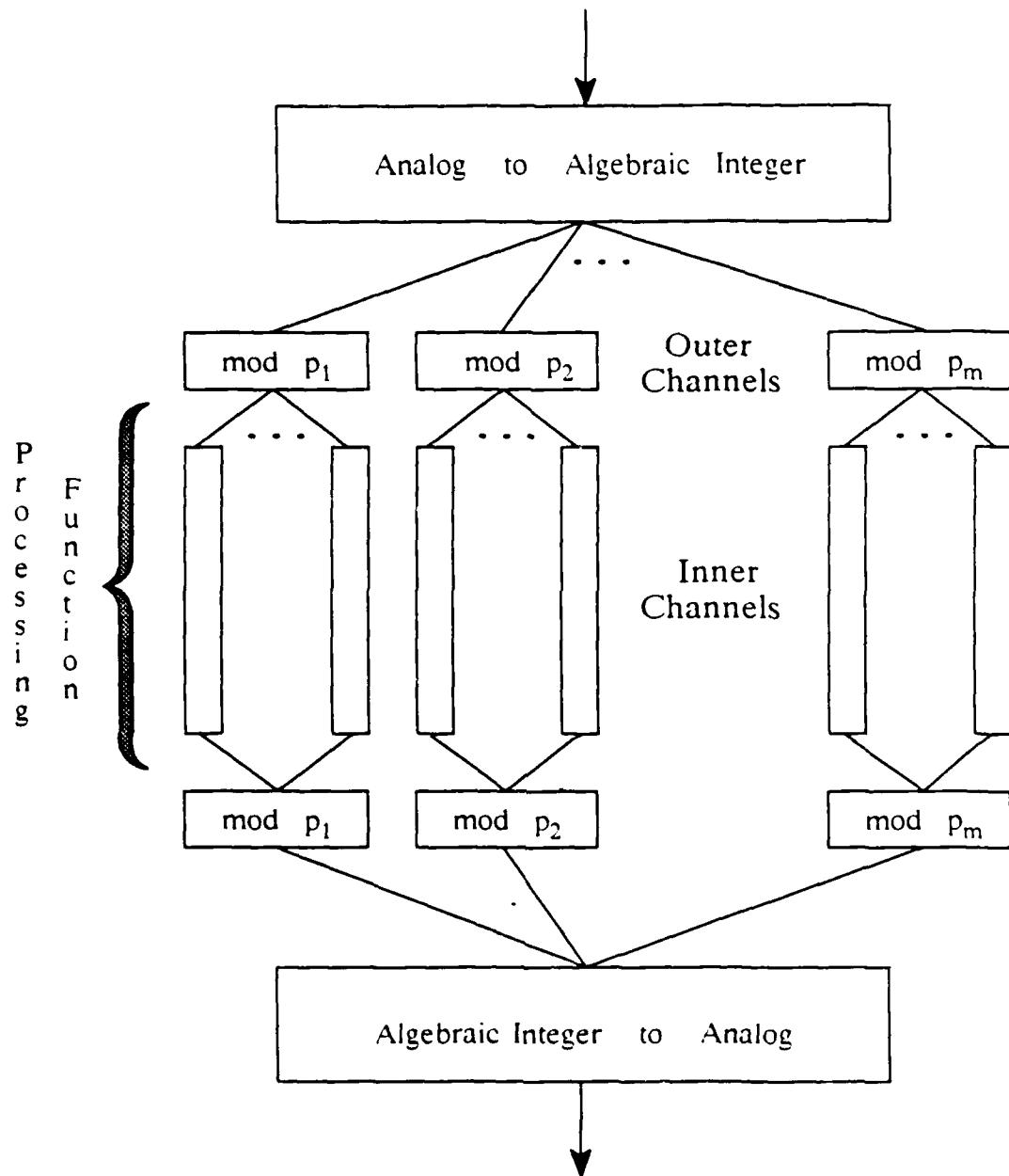


Figure 1. RNS Processing with Algebraic Integers

One objective of this work was to obtain results that were independent of a particular processing function or application. Thus, for example, the performance of quantizers derived from the algebraic integers are characterized, as is common, in terms of signal-to-noise ratios assuming particular probability distributions on the inputs. On the other hand, the impact on overall performance of the *requantization* from algebraic integers back to analog or digital form is expected to be application and processing function dependent. This paper considers the effects of requantization in the case of a processing function consisting of a sum of products, and illustrates the issues involved for a 55-tap finite impulse response filter.

The next section introduces the algebraic-integer and RNS concepts and notations that are needed in this paper. Section 3 treats the problem of analog-to-algebraic-integer conversion. Sections 4 and 5 deal with the two kinds of RNS conversions, which involve, respectively, integer and polynomial RNSs. Section 6 treats the problem of converting back to digital or analog form. Finally, section 7 contains a description of future work and the conclusion.

SECTION 2

THE ALGEBRAIC-INTEGER NUMBER REPRESENTATION AND RESIDUE NUMBER SYSTEM PROCESSING

The idea of using algebraic integers to represent numbers in RNS signal processing can be viewed as a generalization of the quadratic residue number system (QRNS) concept [5], [9]. In this section, the algebraic-integer number representation is introduced. RNS processing is reviewed and combined with the algebraic-integer number representation. A short history of algebraic-integer processing is included.

2.1 ALGEBRAIC-INTEGER EXTENSIONS

Algebraic integers are roots of polynomials that have integer coefficients with the leading coefficient equal to one. For example, $\sqrt{2}$ is an algebraic integer since it is a root of the polynomial $x^2 - 2$. For the present purpose, a general algebraic integer can be viewed as an integer linear combination of certain fixed irrational or imaginary numbers. The introduction that follows uses a sequence of examples; a formal treatment can be found in [8]. The examples involve both real and complex algebraic integers. The real and complex examples are related, and this relationship will be used to reduce the complexity of implementations involving in-phase (I) and quadrature (Q) processing channels. In the following, \mathbf{Z} , \mathbf{Q} , \mathbf{R} , and \mathbf{C} denote, respectively, the integers, the rational numbers, the real numbers, and the complex numbers. The imaginary number $\sqrt{-1}$ is denoted by i .

The example of the *Gaussian integers* $\{a + bi : a, b \in \mathbf{Z}\}$, which are denoted by $\mathbf{Z}[i]$ and form the basis of QRNS, should be kept in mind. They can be viewed as being constructed by adjoining i to the integers. The fact that i satisfies $i^2 = -1$, equivalently that i is a root of $x^2 + 1$, is used to reduce higher powers of i when multiplying elements of $\mathbf{Z}[i]$. This theme is repeated in each of the examples of this section.

2.1.1 Adjoining $\sqrt{2}$

The first example involves *real* algebraic integers. If $\sqrt{2}$ is *adjoined* to the field \mathbf{Q} , the field $\mathbf{Q}(\sqrt{2}) = \{a + b\sqrt{2} : a, b \in \mathbf{Q}\}$ results. The subset of $\mathbf{Q}(\sqrt{2})$ of interest is $\mathbf{Z}[\sqrt{2}] = \{a + b\sqrt{2} : a, b \in \mathbf{Z}\}$. The set $\mathbf{Z}[\sqrt{2}]$ is closed under addition and multiplication:

$$(a + b\sqrt{2}) + (c + d\sqrt{2}) = (a + c) + (b + d)\sqrt{2}, \quad (2.1)$$

$$(a + b\sqrt{2})(c + d\sqrt{2}) = (ac + 2bd) + (ad + bc)\sqrt{2}. \quad (2.2)$$

With these operations, $\mathbb{Z}[\sqrt{2}]$ forms a *ring*. The elements of $\mathbb{Z}[\sqrt{2}]$ are algebraic integers (they are roots of quadratic or linear polynomials with integer coefficients and leading coefficient equal to one). For this example, $\mathbb{Z}[\sqrt{2}]$ contains all the algebraic integers of $\mathbb{Q}(\sqrt{2})$ and is referred to as *the ring of algebraic integers for $\mathbb{Q}(\sqrt{2})$* . The analogy to keep in mind is that $\mathbb{Z}[\sqrt{2}]$ is to $\mathbb{Q}(\sqrt{2})$ as the ring \mathbb{Z} is to the field \mathbb{Q} .

$\mathbb{Q}(\sqrt{2})$ is a *second-degree extension* of \mathbb{Q} since the adjoined element, $\sqrt{2}$, is a root of an irreducible polynomial of degree two over \mathbb{Q} , called its *minimum polynomial*. This relation for $\sqrt{2}$ is actually used in (2.2) to obtain an element of the form $a + b\sqrt{2}$ (in the same way the relation $i^2 = -1$ for i is used in complex arithmetic).

The elements 1 and $\sqrt{2}$ form a basis for $\mathbb{Q}(\sqrt{2})$ as a vector space over \mathbb{Q} . In other words, the elements of $\mathbb{Q}(\sqrt{2})$ are uniquely represented by the form $a + b\sqrt{2}$. Consequently, the elements of $\mathbb{Z}[\sqrt{2}]$ are also represented uniquely as integer linear combinations of 1 and $\sqrt{2}$.

The ring $\mathbb{Z}[\sqrt{2}]$ is in fact a dense subset of \mathbb{R} , meaning that any real number x can be approximated to any desired degree of accuracy by elements of $\mathbb{Z}[\sqrt{2}]$. However, to obtain increasing accuracy, the coefficients a and b grow in size. For example, table 1 lists elements of $\mathbb{Z}[\sqrt{2}]$ with decreasing magnitude and increasing coefficient sizes. Integer multiples of these points can be used to obtain finer and finer grids for approximating elements of \mathbb{R} . These approximations come from the *convergents* of the continued fraction expansion of $\sqrt{2}$, and the list can be continued to obtain any desired degree of accuracy.

Table 1. Elements of $\mathbb{Z}[\sqrt{2}]$ with
Decreasing Magnitude

a	b	$a + b\sqrt{2}$
-1	1	.4142
3	-2	.1716
-7	5	.0711
17	-12	.0294
-41	29	.0122
99	-70	.0051
-239	169	.0021

Algebraic integers can be used to represent the numerical quantities in finite wordlength implementations of signal processing functions. For example, elements of

\mathbb{R} could be represented by ordered pairs (a, b) of integers that correspond to elements $a + b\sqrt{2} \in \mathbb{Z}[\sqrt{2}]$. Equations (2.1) and (2.2) give the rules of addition and multiplication for the ordered-pair representation.

In a practical implementation, the sizes of these coefficients must be constrained. In particular, for an integer M , define the set $\mathbb{Z}[\sqrt{2}]_M \equiv \{a + b\sqrt{2} : a, b \in \mathbb{Z}, |a| \leq M/2, |b| \leq M/2\}$. $\mathbb{Z}[\sqrt{2}]_M$ is a finite set. Figure 2 displays the 25 elements of $\mathbb{Z}[\sqrt{2}]_4$. This case is indicative of the general situation that such finite collections of algebraic integers form nonuniform quantizers, as compared to the usual uniformly spaced quantizers.



Figure 2. $\mathbb{Z}[\sqrt{2}]_4$

2.1.2 Adjoining $e^{2\pi i/8}$

Whereas the last section involved a *real* extension, this section develops a *complex* extension, applicable in the case of signal processing with I and Q channels. This example can be viewed as a generalization of the Gaussian integers $\mathbb{Z}[i]$, the ring of algebraic integers for $\mathbb{Q}(i) = \{a + bi : a, b \in \mathbb{Q}\}$. Let $\omega = e^{2\pi i/8}$ denote the primitive complex eighth root of unity. Since $\omega^4 = -1$, ω is a root of $x^4 + 1$, its minimum polynomial, and ω is an algebraic integer.

If ω is adjoined to \mathbb{Q} , the field $\mathbb{Q}(\omega) = \{a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 : a_i \in \mathbb{Q}, i = 0, 1, 2, 3\}$ results. The representation subset is $\mathbb{Z}[\omega] = \{a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 : a_i \in \mathbb{Z}, i = 0, 1, 2, 3\}$. $\mathbb{Z}[\omega]$ can be viewed geometrically as in figure 3 as containing integer linear combinations of the *basis* vectors 1, ω , ω^2 , and ω^3 . These four vectors do, in fact, form

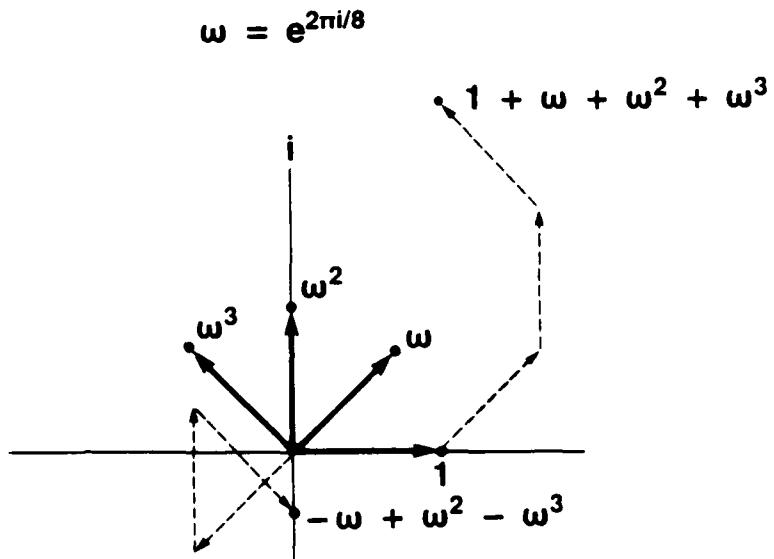


Figure 3. Basis Vectors for $\mathbb{Z}[\omega]$

a vector space basis when the coefficients are restricted to lie in \mathbb{Q} ; that is, the elements of $\mathbb{Z}[\omega]$ are represented uniquely.

Elements of $\mathbb{Z}[\omega]$ can also be viewed as polynomials in *indeterminate* ω with the additional relation that ω satisfies the equation $\omega^4 = -1$. Elements of $\mathbb{Z}[\omega]$ are added and multiplied like polynomials, except, in the case of multiplication, higher powers of ω are reduced using the relation $\omega^4 = -1$.

It is convenient to represent the multiplication of elements of $\mathbb{Z}[\omega]$ in terms of matrices and vectors. Denote the element $a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3$ by the vector (a_0, a_1, a_2, a_3) , and consider the product

$$(a_0, a_1, a_2, a_3) * (b_0, b_1, b_2, b_3) = (c_0, c_1, c_2, c_3).$$

The constant term c_0 is computed by the following cyclic convolution:

$$c_0 = a_0 b_0 - a_1 b_3 - a_2 b_2 - a_3 b_1,$$

which in matrix form is

$$c_0 = (a_0, a_1, a_2, a_3) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}. \quad (2.3)$$

The matrices, which will be called *coefficient forms*, for the coefficients c_1, c_2 , and c_3 are, respectively,

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \quad (2.4)$$

With addition and multiplication defined as above, $\mathbb{Z}[\omega]$ forms a ring and is, in fact, *the* ring of algebraic integers for $\mathbb{Q}(\omega)$.

$\mathbb{Z}[\omega]$ is a dense subset of \mathbb{C} , and so any complex number $x+yi$ can be approximated arbitrarily well by elements of $\mathbb{Z}[\omega]$. As in the first example, for an integer M , define $\mathbb{Z}[\omega]_M \equiv \{a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 : a_i \in \mathbb{Z}, |a_i| \leq M/2, i = 0, 1, 2, 3\}$. $\mathbb{Z}[\omega]_M$ is a finite set and forms a (nonuniform) quantizer of \mathbb{C} . The representatives in $\mathbb{Z}[\omega]_M$ are regarded as vectors (a_0, a_1, a_2, a_3) and are added using ordinary vector addition and are multiplied using (2.3) and (2.4).

It is possible to picture $\mathbb{Z}[\omega]_M$ in \mathbb{R}^2 . If $x+yi = (a_0, a_1, a_2, a_3)$, then since $\omega = \sqrt{2}/2 + i\sqrt{2}/2$,

$$x = a_0 + \frac{\sqrt{2}}{2}(a_1 - a_3), \quad (2.5)$$

$$y = a_2 + \frac{\sqrt{2}}{2}(a_1 + a_3). \quad (2.6)$$

Figure 4 shows the 81 points of $\mathbb{Z}[\omega]_2$. These points display eight-fold rotational symmetry since if $(a_0, a_1, a_2, a_3) \in \mathbb{Z}[\omega]_M$, then

$$\omega(a_0, a_1, a_2, a_3) = (-a_3, a_0, a_1, a_2)$$

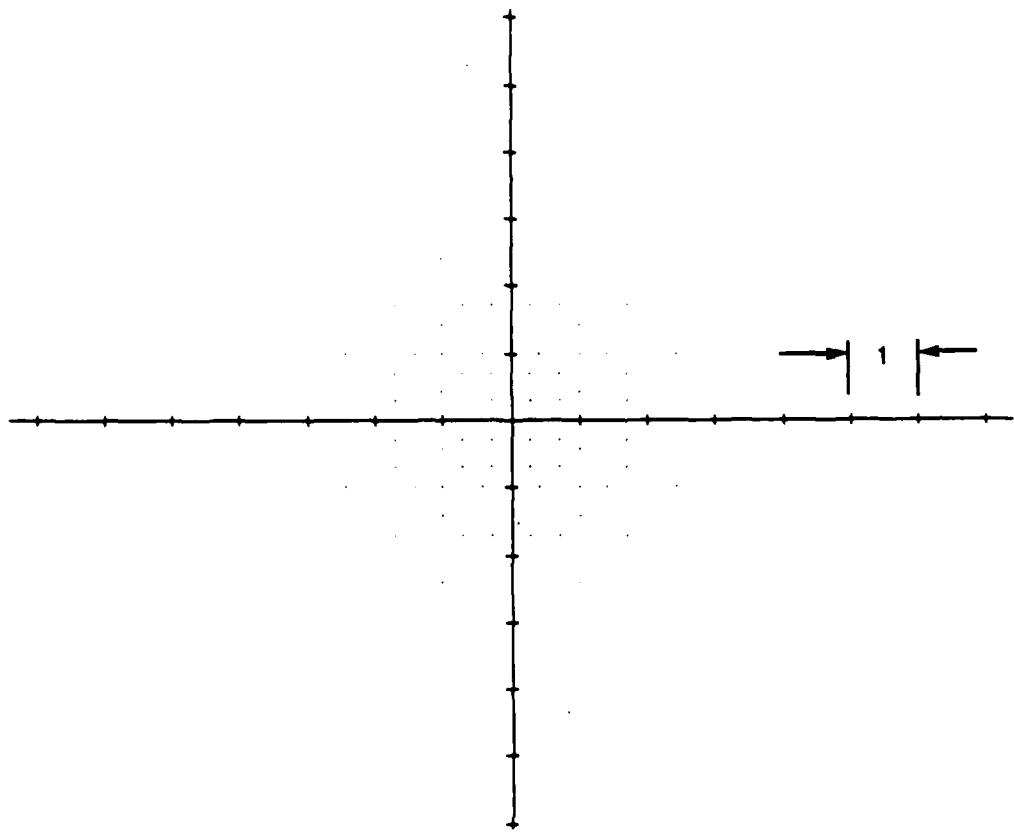


Figure 4. $\mathbb{Z}[e^{2\pi i/8}]_2$

is in $\mathbb{Z}[\omega]_M$.

Finally, there is a relationship between $\mathbb{Z}[\sqrt{2}]$ and $\mathbb{Z}[\omega]$. Consider an element $x + yi = (a_0, a_1, a_2, a_3) \in \mathbb{Z}[\omega] \cap \mathbb{R}$. Since a_1, a_2 , and a_3 are integers, (2.6) implies that a point with $y = 0$ must have $a_2 = 0$ and $a_1 = -a_3$. The expression (2.5) for x becomes

$$x = a_0 - \sqrt{2}a_3.$$

Thus, $\mathbb{Z}[\sqrt{2}]$ corresponds to the real numbers of $\mathbb{Z}[e^{2\pi i/8}]$.

2.1.3 Adjoining $\sqrt{2 + \sqrt{2}}$

This section develops a 4th degree real extension. Let $\theta = \sqrt{2 + \sqrt{2}}$. The minimum polynomial for θ is $x^4 - 4x^2 + 2$. Integer linear combinations of the basis vectors 1, θ , θ^2 , and θ^3 yield $\mathbb{Z}[\theta] = \{a_0 + a_1\theta + a_2\theta^2 + a_3\theta^3 : a_i \in \mathbb{Z}, i = 0, 1, 2, 3\}$. Elements of $\mathbb{Z}[\theta]$ can be viewed as polynomials, and $a_0 + a_1\theta + a_2\theta^2 + a_3\theta^3$ is denoted by (a_0, a_1, a_2, a_3) .

When elements of $\mathbb{Z}[\theta]$ are multiplied, the relation $\theta^4 = 4\theta^2 - 2$ is used to reduce powers of θ above three. The coefficient forms for a product (c_0, c_1, c_2, c_3) are, respectively,

$$\begin{aligned} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \\ 0 & -2 & 0 & -8 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 \end{pmatrix}, \\ & \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 4 \\ 1 & 0 & 4 & 0 \\ 0 & 4 & 0 & 14 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 4 \\ 1 & 0 & 4 & 0 \end{pmatrix}. \end{aligned} \quad (2.7)$$

For example,

$$(0, 0, 0, 1) * (0, 0, 0, 1) = (-8, 0, 14, 0). \quad (2.8)$$

For an integer M , define the set $\mathbb{Z}[\theta]_M \equiv \{a_0 + a_1\theta + a_2\theta^2 + a_3\theta^3 : a_i \in \mathbb{Z}, |a_i| \leq M/2, i = 0, 1, 2, 3\}$. $\mathbb{Z}[\theta]_M$ is a finite set and forms a nonuniform quantizer for \mathbb{R} .

An algebraic integer can often be represented using more than one basis. This is demonstrated for $\mathbb{Z}[\theta]$. The usual polynomial basis, $\{1, \theta, \theta^2, \theta^3\}$ will be denoted by \mathbf{B}_1 . The number $\sqrt{2 - \sqrt{2}}$ is in $\mathbb{Z}[\theta]$ since

$$\sqrt{2 - \sqrt{2}} = -3\sqrt{2 + \sqrt{2}} + (\sqrt{2 + \sqrt{2}})^3.$$

Let $\mathbf{B}_2 = \{1, \sqrt{2 + \sqrt{2}}, \sqrt{2}, \sqrt{2 - \sqrt{2}}\}$; then $\mathbb{Z}[\theta]$ can be represented also in terms of the elements of \mathbf{B}_2 .

This is most easily seen in terms of the change of basis matrix. Let (a_0, a_1, a_2, a_3) correspond to $a_0 + a_1\theta + a_2\theta^2 + a_3\theta^3$, and let (b_0, b_1, b_2, b_3) correspond to $b_0 + b_1\sqrt{2 + \sqrt{2}} + b_2\sqrt{2} + b_3\sqrt{2 - \sqrt{2}}$. The two representations are then related by:

$$(b_0, b_1, b_2, b_3) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & -3 & 0 & 1 \end{pmatrix} = (a_0, a_1, a_2, a_3),$$

$$(b_0, b_1, b_2, b_3) = (a_0, a_1, a_2, a_3) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{pmatrix}. \quad (2.9)$$

In general, an invertible integer matrix corresponds to a suitable change of basis matrix exactly when the inverse matrix contains only integers.

There are two general remarks that apply to representing algebraic integers using different bases. When coefficient sizes are restricted, the finite set $\mathbb{Z}[\theta]_M$ obtained depends on the choice of basis. In this example, since the elements of \mathbf{B}_2 are closer together in magnitude than the elements of \mathbf{B}_1 , there are more points clustered around zero for the \mathbf{B}_2 basis. This may be an advantage depending on the density of the inputs being quantized.

Secondly, the dynamic range growth that results when two elements of $\mathbb{Z}[\theta]$ are multiplied depends on the basis representation. In this example, the coefficient forms for a product (c_0, c_1, c_2, c_3) expressed using the \mathbf{B}_2 basis are, respectively,

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{pmatrix}. \quad (2.10)$$

Now, for example, (comparing to (2.8))

$$(0, 0, 0, 1) * (0, 0, 0, 1) = (2, 0, -1, 0), \quad (2.11)$$

and \mathbf{B}_2 yields the preferred representation from this standpoint.

2.1.4 Adjoining $e^{2\pi i/16}$

This final example involves an 8th degree complex extension that will turn out to intersect \mathbf{R} at $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$. Let $\omega = e^{2\pi i/16}$ denote the primitive complex 16th root of unity. The minimum polynomial of ω is $x^8 + 1$. This example is very similar to the previous $\mathbb{Z}[e^{2\pi i/8}]$ example.

$\mathbb{Z}[\omega]$ can be regarded as consisting of polynomials of degree seven with integer coefficients. The rule $\omega^8 = -1$ is used in the product to reduce the degree of powers of ω above seven. If $x + yi = (a_0, a_1, \dots, a_7) \in \mathbb{Z}[\omega]$, since $\omega = \sqrt{2 + \sqrt{2}}/2 + (\sqrt{2 - \sqrt{2}}/2)i$, then

$$\begin{aligned} x = a_0 + (\sqrt{2 + \sqrt{2}}/2)(a_1 - a_7) + (\sqrt{2}/2)(a_2 - a_6) \\ + (\sqrt{2 - \sqrt{2}}/2)(a_3 - a_5), \end{aligned} \quad (2.12)$$

$$y = a_4 + (\sqrt{2 + \sqrt{2}}/2)(a_3 + a_5) + (\sqrt{2}/2)(a_2 + a_6) + (\sqrt{2 - \sqrt{2}}/2)(a_1 + a_7). \quad (2.13)$$

Figure 5 shows the 6,561 points of $\mathbb{Z}[\omega]_2$. These points have 16-fold rotational symmetry.

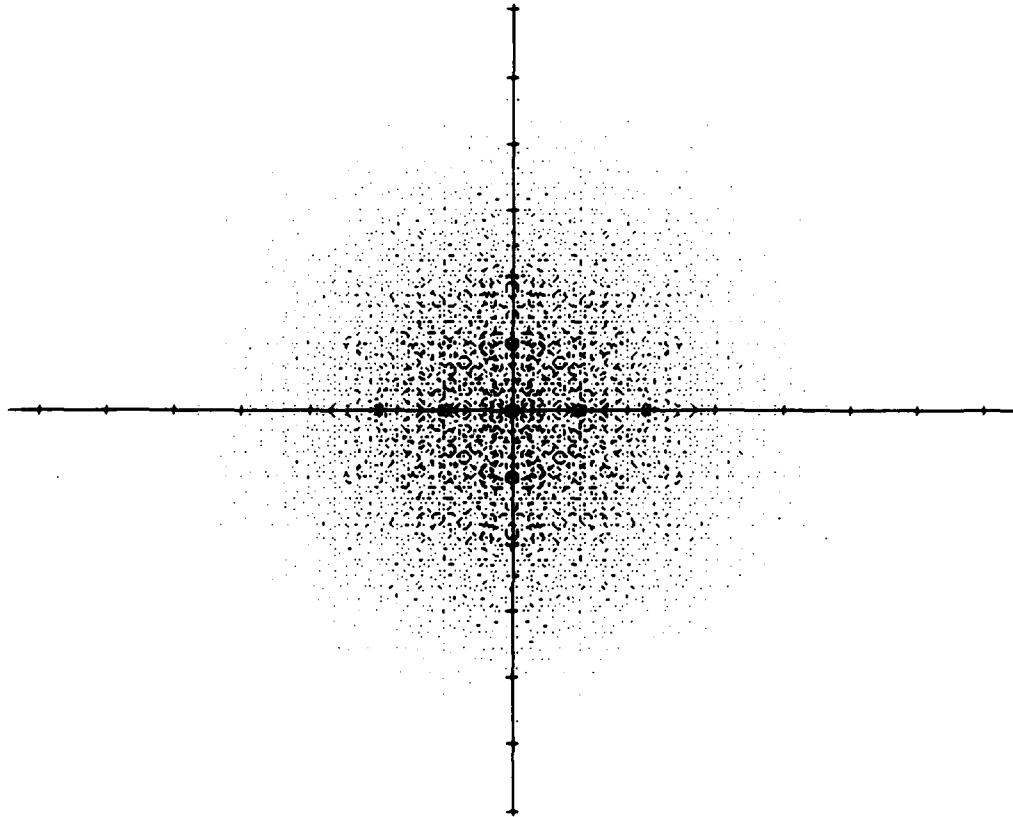


Figure 5. $\mathbb{Z}[e^{2\pi i/16}]_2$

Finally, consider an element $x + yi = (a_0, a_1, \dots, a_7) \in \mathbb{Z} \cap \mathbb{R}$. Since the coefficients are integers, (2.13) implies a point with $y = 0$ must have $a_4 = 0$, $a_7 = -a_1$, $a_6 = -a_2$, and $a_5 = -a_3$. The expression (2.12) for x becomes

$$x = a_0 + \sqrt{2 + \sqrt{2}} a_1 + \sqrt{2} a_2 + \sqrt{2 - \sqrt{2}} a_3. \quad (2.14)$$

Thus, $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$, expressed using the $\mathbf{B}_2 = \{1, \sqrt{2 + \sqrt{2}}, \sqrt{2}, \sqrt{2 - \sqrt{2}}\}$ basis, corresponds to the real numbers of $\mathbb{Z}[e^{2\pi i/16}]$.

2.2 RESIDUE NUMBER SYSTEM PROCESSING WITH ALGEBRAIC INTEGERS

In most digital arithmetic circuits there is an inherent trade-off between speed and precision of calculation due to factors such as carry propagation, for example. One way around this problem is to perform the computations using a residue number system (RNS). In this approach, high-dynamic-range integer processing is performed in a number of parallel low-dynamic-range channels. For a comprehensive introduction to RNS digital signal processing, including reprints of certain key papers, see [19]. This section briefly introduces ordinary integer RNS and shows how these ideas extend to the algebraic-integer representation.

2.2.1 Integer RNS Processing

A residue number system is defined by n positive integers m_i that are all relatively prime (that is, the greatest common divisor of any pair of them is one). The integers m_i are referred to as the system *moduli*, and the number M ,

$$M = \prod_{i=1}^n m_i,$$

is called the system *range*. Given an RNS and an integer X , the Chinese Remainder Theorem (CRT) states that X is represented uniquely up to a multiple of M by its *residues* x_i ,

$$x_i = \langle X \rangle_{m_i}, \quad i = 1, 2, \dots, n,$$

where the operation $\langle X \rangle_m$, read X *modulo* m , indicates taking the nonnegative remainder of the quantity X divided by m . As a consequence of the CRT, then any integer X between 0 and $M - 1$, inclusive, can be reconstructed from its residues x_i .

For an integer m , the residues modulo m form the set $\{0, 1, \dots, m - 1\}$, which is denoted by \mathbb{Z}_m . \mathbb{Z}_m forms a ring with addition and multiplication defined for $a, b \in \mathbb{Z}_m$ by:

$$a + b = \langle a + b \rangle_m,$$

$$ab = \langle ab \rangle_m.$$

Some other notation: for an integer X and $a \in \mathbb{Z}_p$, $X \equiv a \pmod{m}$ means $\langle X \rangle_m = a$.

In integer RNS processing, the numerical quantities are quantized to integers, perhaps by scaling and rounding, and reduced modulo the system moduli. The same calculation, consisting of additions and multiplications, is performed independently in parallel channels, one for each of the system moduli. The CRT is then used to reconstruct the result, which is correct provided the actual answer is smaller than the

system range. The limiting factors on speed are usually the input and output conversion processes, where the system goes into and comes out of the residue representation.

2.2.2 Algebraic-Integer RNS Processing

An algebraic integer may be naturally represented in several ways. The choice of representation depends on a number of factors, for example, on the analog signals being quantized, or on the need to minimize the dynamic range growth due to multiplication. In order to make this discussion more concrete, it is assumed that the algebraic integers are of the form

$$a_0 + a_1\theta + a_2\theta^2 + \cdots + a_{n-1}\theta^{n-1}, \quad (2.15)$$

where θ has a minimum polynomial $f(x)$ of degree n . Other algebraic-integer representations are considered in section 5.

Let the RNS consist of the primes p_1, p_2, \dots, p_m . This restriction is being made to ease the exposition; generalizations to arbitrary relatively prime moduli are pointed out in section 5. RNS processing with algebraic integers was depicted schematically in figure 1. After analog signals are converted to algebraic integers, each coefficient of the algebraic integer is reduced modulo the primes in the RNS. Since the coefficients of the algebraic-integer representations are likely to be smaller than the primes in the RNS, the reduction modulo p_i will usually just amount to changing the negative coefficients from, say, 2's-complement to p_i 's-complement form. This first modulo reduction, corresponding to integer RNS, yields the so-called *outer level of parallelism*.

In each modulo p outer channel, the algebraic integer (2.15) is represented by the vector of residues

$$(\langle a_0 \rangle_p, \langle a_1 \rangle_p, \dots, \langle a_{n-1} \rangle_p). \quad (2.16)$$

While addition of these representatives is performed coefficient-by-coefficient in n independent parallel channels, the multiplication involves a convolution that requires *cross-talk* between the channels (see the coefficient forms of the examples in section 2.1). It is possible, however, to choose the prime p so that there exists what amounts to a number-theoretic transform that can be applied to (2.16) to produce a vector of n residues modulo p . These transformed representatives can now be added and multiplied by considering each coefficient independently, and processing is accomplished by n independent modulo p channels; this is the *inner level of parallelism*, which is depicted in figure 1.

The polynomial version of the CRT provides the theoretical basis for the inner level of parallelism in algebraic-integer RNS. The representative (2.16) is viewed as a polynomial over \mathbb{Z}_p ; that is, an element of the ring $\mathbb{Z}_p[x]$. But more is true: the indeterminate θ satisfies the relation $f(\theta) = 0$, so that (2.16) can be regarded as an

element of the residue ring $\mathbf{Z}_p[x]$ modulo $f(x)$, which is denoted by $\mathbf{Z}_p[x]/f(x)$. Now the polynomial version of the CRT involving the factors of $f(x)$ can be applied; section 5 gives the complete treatment.

It is useful to state here the condition on the prime p that guarantees the existence of the inner level of parallelism, namely, that the prime p should be chosen so that the polynomial $f(x)$ factors completely into linear factors over \mathbf{Z}_p . For example, for $\mathbf{Z}[\sqrt{2}]$, $f(x) = x^2 - 2$ and \mathbf{Z}_p should contain a " $\sqrt{2}$ ". It can easily be checked that $p = 7, 17, 23, 31, 41, 47, 71, 73, 79, 89, 97, 103, 113, 119$, and 127 are the suitable primes with seven bits or less. For $\mathbf{Z}[e^{2\pi i/8}]$ and $\mathbf{Z}[e^{2\pi i/16}]$, the condition on p is, respectively, $p \equiv 1 \pmod{8}$ and $p \equiv 1 \pmod{16}$. For Gaussian integers $\mathbf{Z}[i]$, the condition is $p \equiv 1 \pmod{4}$, corresponding to the familiar case of QRNS.

2.3 HISTORY OF ALGEBRAIC-INTEGER PROCESSING

The idea of using algebraic integers in RNS signal processing can be traced to earlier work on QRNS. Motivated by work on QRNS, which was eventually published in [5], Cozzens and Finkelstein, in January 1983, generalized this idea to higher-degree cyclotomic extensions. Their paper [3] focused on using $\mathbf{Z}[e^{2\pi i/2^r}]$, cyclotomic extensions of degree a power of two, to compute the discrete Fourier transform. A subsequent paper [4] gave a more complete error and range analysis for this case, including a methodology for computing an achievable upper bound, called the *exact bound*, on the range of the coefficients in the case of a two-stage quantization scheme, which is given in detail in section 3.2.

It was clear from the outset that, in addition to having nice algebraic properties, the algebraic integers possessed a rich quantization structure. The paper [6] considered the algebraic-integer approximation problem, deriving error estimates for the case of $\mathbf{Z}[e^{2\pi i/8}]$ in a region near the origin. A subsequent paper, [7], outlined an approximation algorithm that was implicit in the proofs in [6]. Both papers made the point that to obtain *efficient* algebraic-integer approximations, scaling, as in the conventional approach, and direct algebraic-integer approximations had to be combined. The paper [14] developed an approximation routine for $\mathbf{Z}[e^{2\pi i/8}]$ by treating real and imaginary parts separately, an approach similar to the one in section 3.3.

Algebraic-integer number representations were used in [1] ($\mathbf{Z}[\sqrt{2}]$) to design a two-dimensional discrete cosine transform, and in [2] ($\mathbf{Z}[e^{2\pi i/16}]$) to compute the numerically approximate inverse of a square matrix defined over the complex numbers. The paper [1] also independently derived the exact bound methodology.

2.4 CONCLUSION

The algebraic-integer number system was introduced using four examples: $\mathbb{Z}[\sqrt{2}]$, $\mathbb{Z}[e^{2\pi i/8}]$, $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$, and $\mathbb{Z}[e^{2\pi i/16}]$. In general, the cyclotomic extensions $\mathbb{Z}[e^{2\pi i/2^r}]_M$ have rotational symmetries not enjoyed by other complex extensions, small multiplicative dynamic range growth, and representatives with 2^{r-1} coefficients—all features that make these extensions attractive for signal processing applications involving I and Q channels. The real extensions $\mathbb{Z}[\sqrt{2}]$ and $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$ can be obtained from $\mathbb{Z}[e^{2\pi i/8}]$ and $\mathbb{Z}[e^{2\pi i/16}]$ by restricting to the real line, a fact that will prove useful in section 3. Although many other extensions are possible, the four examples introduced are expected to play an important role in future algebraic-integer implementations.

The algebraic-integer representation was combined with RNS processing. For each prime p of a suitably chosen RNS, the ordinary outer channel corresponding to p splits into parallel modulo p inner channels, one for each coefficient in the algebraic-integer representation. Besides yielding an increase in RNS parallelism, this further splitting makes the complicated algebraic-integer multiplication easy to compute in parallel.

SECTION 3

ANALOG-TO-ALGEBRAIC-INTEGER CONVERSION

This section considers the problem of representing analog values in terms of algebraic integers. There are two types of values that need to be represented: constants that are known beforehand, such as the coefficients of a time-invariant filter, and the varying inputs to a real-time processor. The former do not present as much of a problem since they can be approximated beforehand using exhaustive computer routines. The problem involving the latter type is the subject of this section.

Two approaches are considered. The first is a direct analog-to-algebraic-integer conversion for real algebraic integers. This approach (actually two methods are suggested) uses nonlinear functions, making actual hardware implementations difficult. The second approach, which also applies in the case of real algebraic integers and represents an engineering compromise, uses a (conventional) uniform analog-to-digital (A/D) converter followed by a precomputed approximation table, which performs a digital-to-algebraic-integer conversion. (The digital-to-algebraic-integer conversion may be all that is required of an algebraic-integer processor that is embedded in a conventional digital processor.)

The complexity of the problem grows if the inputs are complex, and a strategy that treats separate I (real) and Q (imaginary) channels is described. Finally, the performance of the different algebraic-integer quantizers is evaluated for the case of $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$.

3.1 DIRECT ANALOG-TO-ALGEBRAIC-INTEGER CONVERSION

In this section, all algebraic integers considered are real. Two methods for direct analog-to-algebraic-integer conversion are presented. The first uses the *compressor characteristic* of the nonuniform algebraic-integer quantizer. The second is similar to a conventional successive approximation A/D converter.

3.1.1 Compressor Characteristic Method

Finite sets of real algebraic integers with bounded coefficient sizes generally correspond to nonuniform quantizers of \mathbb{R} . In general, nonuniform quantization can be achieved by compressing the signal by a nonuniform compressor characteristic c , by quantizing the compressed signal with a uniform quantizer, and then applying the inverse c^{-1} to the quantized signal. The compressor characteristic c is also called the *companding law* (for *compressing* and *expanding*) [10].

Figure 6 shows the limiting values of the compressor characteristic for the sets $\mathbb{Z}[\sqrt{2}]_M$ and $\mathbb{Z}[\sqrt{2} + \sqrt{2}]_M$ (using the nonpolynomial B_2 basis) as the value of M tends toward infinity. Only the positive values are shown (scaled to the interval $[0,1]$); the negative values are obtained by reflection through the origin. The curve for $\mathbb{Z}[\sqrt{2}]$ was obtained analytically by considering the density of points in $\mathbb{Z}[\sqrt{2}]$. It consists of a linear part near the origin and a quadratic part beginning at the point $(\sqrt{2} - 1)^2$. The curve for $\mathbb{Z}[\sqrt{2} + \sqrt{2}]$ was obtained by computer (a similar analysis for this case is possible but is much more complicated).

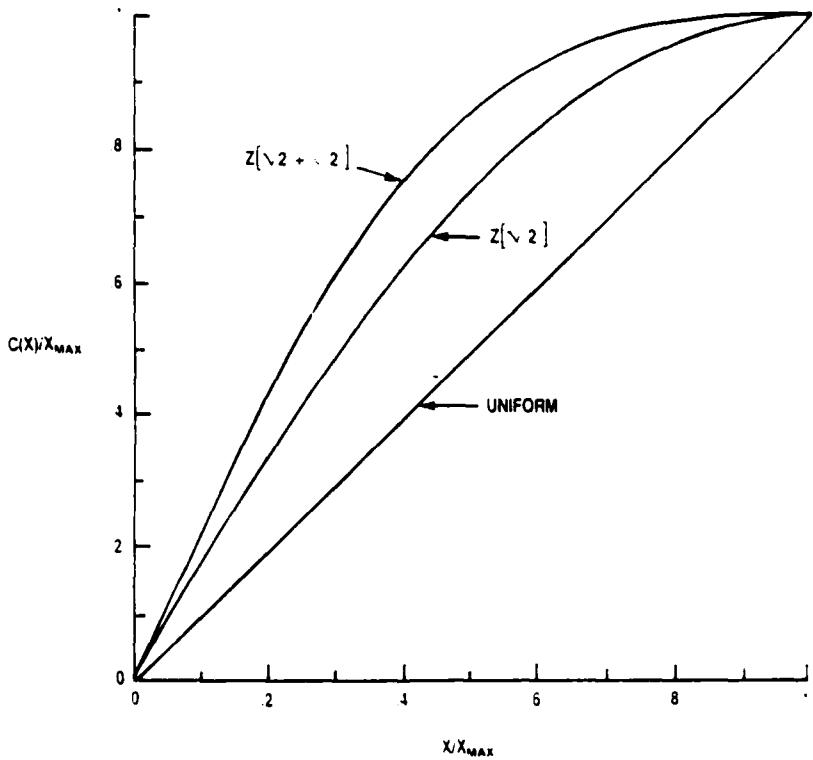


Figure 6. Compressor Characteristics for Real Algebraic Integers

The compressor characteristic can be used to obtain a direct analog-to-algebraic-integer converter. The converter is based on the following property: if the points of $\mathbb{Z}[\sqrt{2}]_M$, for instance, are scaled so that the largest value becomes 1, then the compressor characteristic for $\mathbb{Z}[\sqrt{2}]$ maps these points to (nearly) uniformly spaced points between -1 and 1 on the y -axis. A uniform A/D converter is constructed using these representation levels, except, instead of associating the usual digital code with each level, the corresponding algebraic-integer code is used. An analog input, scaled to lie in the interval $[-1, 1]$, is quantized by first applying the compressor characteristic to it and then using the uniform A/D converter on the compressed signal.

Although a nonlinear function has been applied to the signal, linear signal processing (without undesirable intermodulation products beyond those that result from the quantization error) is still possible. This is because the algebraic-integer codes are being used, which has the same effect as applying the inverse of the nonlinearity to the signal. This is in contrast to the situation where a nonlinear function is applied to the signal, such as the μ -law companders used in speech, and the binary codes from the uniform A/D converter are used directly. In this case, linear signal processing with these codes would produce unacceptable intermodulation products.

Implementing accurately in hardware such nonlinear compressor functions is difficult, and it is now more common to use a piecewise linear approximation [10]. The next section gives another strategy for direct analog-to-algebraic-integer conversion that also uses nonlinear functions, but in this case all that is required is that the zero-crossings are located correctly. This may prove to be easier.

3.1.2 Successive Approximation Method

It is possible to translate the real algebraic-integer quantization problem into a form that more closely resembles conventional quantization. For simplicity, this is demonstrated in the case of $\mathbb{Z}[\sqrt{2}]$. Suppose, for example, the coefficients a and b of $a + b\sqrt{2} \in \mathbb{Z}[\sqrt{2}]$ are 2-bit integers in 2's-complement form; that is, $a = -2V_{a1} + V_{a0}$ and $b = -2V_{b1} + V_{b0}$, where $V_{xi} = 0, 1$ (so $-2, -1, 0$, and 1 are represented respectively by $V_{x1}V_{x0} = 10, 11, 00$, and 01). Note that the 2's-complement form for the coefficients yields an asymmetric representation set, which is denoted by $\mathbb{Z}[\sqrt{2}]_{\{-2, -1, 0, 1\}}$.

Then $a + b\sqrt{2}$ can be rewritten as

$$-2V_{a1} + V_{a0} - 2\sqrt{2}V_{b1} + \sqrt{2}V_{b0}. \quad (3.1)$$

Arranging the constants in descending magnitude, and defining V_3 , V_2 , V_1 , and V_0 appropriately, (3.1) becomes

$$-2\sqrt{2}V_3 - 2V_2 + \sqrt{2}V_1 + V_0. \quad (3.2)$$

Table 2. Binary Codes for $\mathbb{Z}[\sqrt{2}]_{\{-2,-1,0,1\}}$

Binary	Decimal	Decision Thresholds
0011	$1 + \sqrt{2} \approx 2.414$	$(1 + 2\sqrt{2})/2, \infty$
0010	$\sqrt{2} \approx 1.414$	$(1 + \sqrt{2})/2, (1 + 2\sqrt{2})/2$
0001	1	$\sqrt{2}/2, (1 + \sqrt{2})/2$
0111	$-1 + \sqrt{2} \approx .414$	$(-1 + \sqrt{2})/2, \sqrt{2}/2$
0000	0	$(1 - \sqrt{2})/2, (-1 + \sqrt{2})/2$
1011	$1 - \sqrt{2} \approx -.414$	$-1/2, (1 - \sqrt{2})/2$
0110	$-2 + \sqrt{2} \approx -.586$	$(-3 + \sqrt{2})/2, -1/2$
0101	-1	$(-1 - \sqrt{2})/2, (-3 + \sqrt{2})/2$
1010	$-\sqrt{2} \approx -1.414$	$(1 - 3\sqrt{2})/2, (-1 - \sqrt{2})/2$
1001	$1 - 2\sqrt{2} \approx -1.828$	$(-1 - 2\sqrt{2})/2, (1 - 3\sqrt{2})/2$
0100	-2	$(-3 - \sqrt{2})/2, (-1 - 2\sqrt{2})/2$
1111	$-1 - \sqrt{2} \approx -2.414$	$(-1 - 3\sqrt{2})/2, (-3 - \sqrt{2})/2$
1000	$-2\sqrt{2} \approx -2.828$	$(-2 - 3\sqrt{2})/2, (-1 - 3\sqrt{2})/2$
1110	$-2 - \sqrt{2} \approx -3.414$	$(-3 - 3\sqrt{2})/2, (-2 - 3\sqrt{2})/2$
1101	$-1 - 2\sqrt{2} \approx -3.828$	$(-3 - 4\sqrt{2})/2, (-3 - 3\sqrt{2})/2$
1100	$-2 - 2\sqrt{2} \approx -4.828$	$-\infty, (-3 - 4\sqrt{2})/2$

Table 2 lists the 16 possible codes for $V_3V_2V_1V_0$, the respective decimal values, and the respective decision thresholds (which occur at the midpoints between successive representation values—the optimal placement [10]).

It is useful to compare the representation in (3.2) with the 4-bit 2's-complement representation

$$-2^3V_3 + 2^2V_2 + 2V_1 + V_0. \quad (3.3)$$

The magnitude of the weights in (3.2) and (3.3) are respectively $\{2\sqrt{2}, 2, \sqrt{2}, 1\}$ and $\{2^3, 2^2, 2, 1\}$. Whereas the absolute values of the weights in (3.3) are *superincreasing* ($2 > 1$, $2^2 > 2 + 1$, and $2^3 > 2^2 + 2 + 1$), the weights in (3.2) are not. This fact accounts for the density of the elements in $\mathbb{Z}[\sqrt{2}]_{\{-2,-1,0,1\}}$.

The structure that performs the conversion for $\mathbb{Z}[\sqrt{2}]_{\{-2,-1,0,1\}}$ is shown in figure 7. The analog input voltage x enters at the top and is applied to each of the four columns. In the *offset level*, the voltage is offset as a function of the 0-1 values of V_3 , V_2 , V_1 , and V_0 by the weights indicated. For example, if $(V_3, V_2, V_1, V_0) = (1, 0, 0, 0)$, then $x + 2\sqrt{2}$ is input into the next level, called the *functional level*. The values of the functions g_3 , g_2 , g_1 , and g_0 will be derived subsequently. Each device in the functional level evaluates

the indicated function at the input voltage and outputs the result to the *threshold level*. Each device in the threshold level outputs the value 0 or 1 depending on whether the input is negative or not. These values, which correspond to the V_i , are fed back to the offset level.

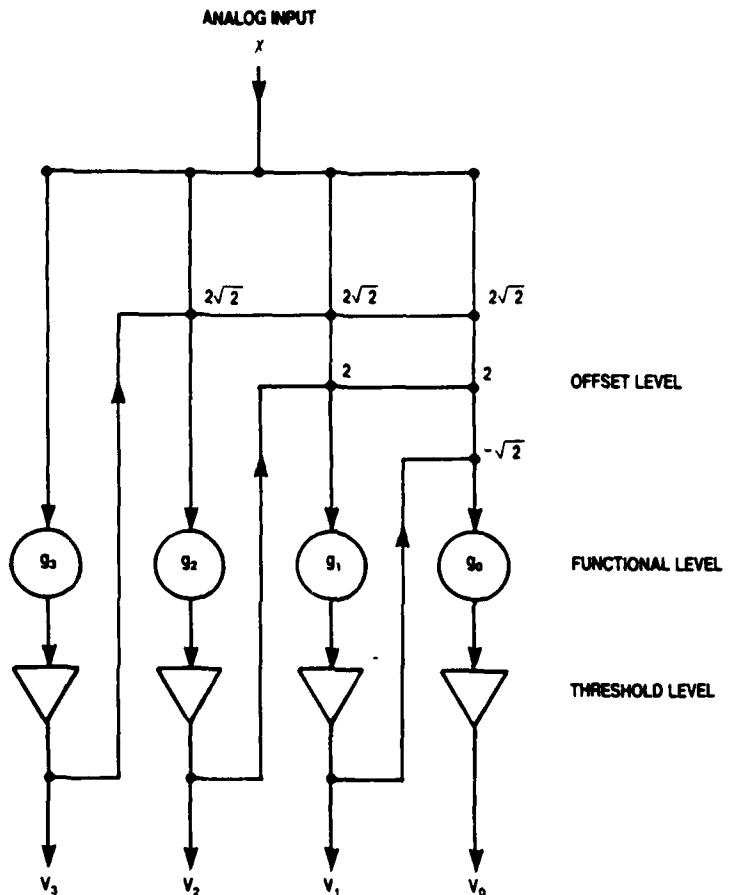


Figure 7. Analog-to-Z $[\sqrt{2}]_{\{-2,-1,0,1\}}$ Converter

The converter operates like a conventional successive approximation A/D converter, except that the threshold decisions are complicated by the fact that the weights are not superincreasing. This complication is embodied in the functions g_i , which are now derived. The V_3 output only depends on the values of the function g_3 . Thus, the function g_3 should be positive on those intervals where $V_3 = 1$ and negative on those intervals where $V_3 = 0$. The following 5th degree polynomial with this property can be

obtained by inspecting table 2:

$$g_3(x) = -(x - (-3 - \sqrt{2})/2)(x - (-1 - 2\sqrt{2})/2)(x - (-1 - \sqrt{2})/2)(x - (-1/2))(x - (1 - \sqrt{2})/2). \quad (3.4)$$

The roots of g_3 correspond to decision thresholds where V_3 changes value.

Once the value of V_3 is determined, then a 3-bit representation for $x + 2\sqrt{2}V_3$ involving the weights -2 , $\sqrt{2}$, and 1 needs to be determined. If $V_3 = 0$, then the input x is passed unaffected to the 3-bit converter (the part in figure 7 corresponding to V_2 , V_1 , and V_0). If, on the other hand, $V_3 = 1$, then $x + 2\sqrt{2}$ is passed to the 3-bit converter. In any case, the 3-bit converter determines V_2 , V_1 , and V_0 so that

$$x + 2\sqrt{2}V_3 = -2V_2 + \sqrt{2}V_1 + V_0, \quad (3.5)$$

determining the element of $\mathbb{Z}[\sqrt{2}]_{\{-2, -1, 0, 1\}}$.

The 3-bit converter is constructed in a similar fashion using the weights -2 , $\sqrt{2}$, and 1 . Again, these weights are not superincreasing and the function g_2 is nonlinear. The polynomial

$$g_2(x) = -(x - (-2 + \sqrt{2})/2)(x - (-1 + \sqrt{2})/2)(x - \sqrt{2}/2) \quad (3.6)$$

has the correct properties; that is, $g_2(x)$ is positive when $V_2 = 1$, and $g_2(x)$ is negative when $V_2 = 0$. It was derived in the same way g_3 was, except that only the weights -2 , $\sqrt{2}$, and 1 were used. Now the offset to the 2-bit converter is 0 or 2 depending on whether $V_2 = 0$ or 1.

Continuing in this fashion, g_1 and g_0 are determined to be:

$$g_1(x) = 2x - (1 + \sqrt{2}), \quad (3.7)$$

$$g_0(x) = 2x - 1, \quad (3.8)$$

with an offset of $-\sqrt{2}$ from the 2-bit to the 1-bit converter.

With these definitions of g_3 , g_2 , g_1 , and g_0 , the converter in figure 7 obtains the correct $\mathbb{Z}[\sqrt{2}]_{\{-2, -1, 0, 1\}}$ representative for any analog input x . The structure can be clocked or run asynchronously—the analog input is applied and the network is allowed to settle.

The accuracy of the converter does not depend on implementing the exact shape of the functions g_i ; it depends just on implementing the zero-crossings of the functions correctly. Thus, the values of the g_i can be *driven to the rails* and be replaced with $\pm C$, for a constant C . Consequently, the accuracy would depend in large part on the

ability to accurately implement these *characteristic functions*. One possible approach would be to use circuits similar to those used in a *window comparator*. Figure 8 plots g_3, g_2, g_1 , and g_0 after hard-limiting for the $\mathbb{Z}[\sqrt{2}]_{\{-2,-1,0,1\}}$ example.

Although the successive approximation method was illustrated for the set $\mathbb{Z}[\sqrt{2}]_{\{-2,-1,0,1\}}$, it can be applied to other cases. However, the complexity grows rapidly as either the number of bits or the degree of the extension is increased. The number of zero-crossings required by each function in the functional level is a good measure of the complexity of the method. The example had complexity (5, 3, 1, 1). No attempt was made to evaluate the complexity of the method in general.

3.2 TWO-STAGE ANALOG-TO-ALGEBRAIC-INTEGER CONVERSION

An analog-to-algebraic-integer converter can be constructed using a conventional A/D converter that is followed by a table of real algebraic-integer approximations. A conventional A/D converter is selected that meets the speed requirements of the processor, and that has more accuracy than the processor would ever require. The uniformly spaced representation values, corresponding to the digital codes, are approximated by algebraic integers beforehand, using exhaustive computer routines. These approximations are stored in tables, which are accessed by the digital codes.

Let $\mathbb{Z}[\theta]$ denote the set of algebraic-integer representatives. Increasing levels of accuracy are achieved by approximating the uniform representation values using the points of $\mathbb{Z}[\theta]_M$ for increasing values of the integer M . Each element of $\mathbb{Z}[\theta]_M$ could be associated with many digital codes (if M is small) or none at all (for larger M). The ultimate accuracy is of course limited by the accuracy of the A/D converter.

This approach is straightforward, but yields suboptimal quantization-error performance. For a fixed integer M , the representation values of a two-step quantizer form a subset of $\mathbb{Z}[\theta]_M$. The decision thresholds of the quantizer, however, are those of the uniform quantizer implemented by the A/D converter. These decision thresholds will certainly not be located at the midpoints of the intervals between successive representation values, which is a necessary condition for optimality. Compare this to the direct algebraic-integer quantizer, where the representation values consist of all the elements of $\mathbb{Z}[\theta]_M$, and the decision thresholds are set at the midpoints between successive representation values. Section 3.4 quantifies the loss in performance for the case of $\mathbb{Z}[\sqrt{2} + \sqrt{2}]$.

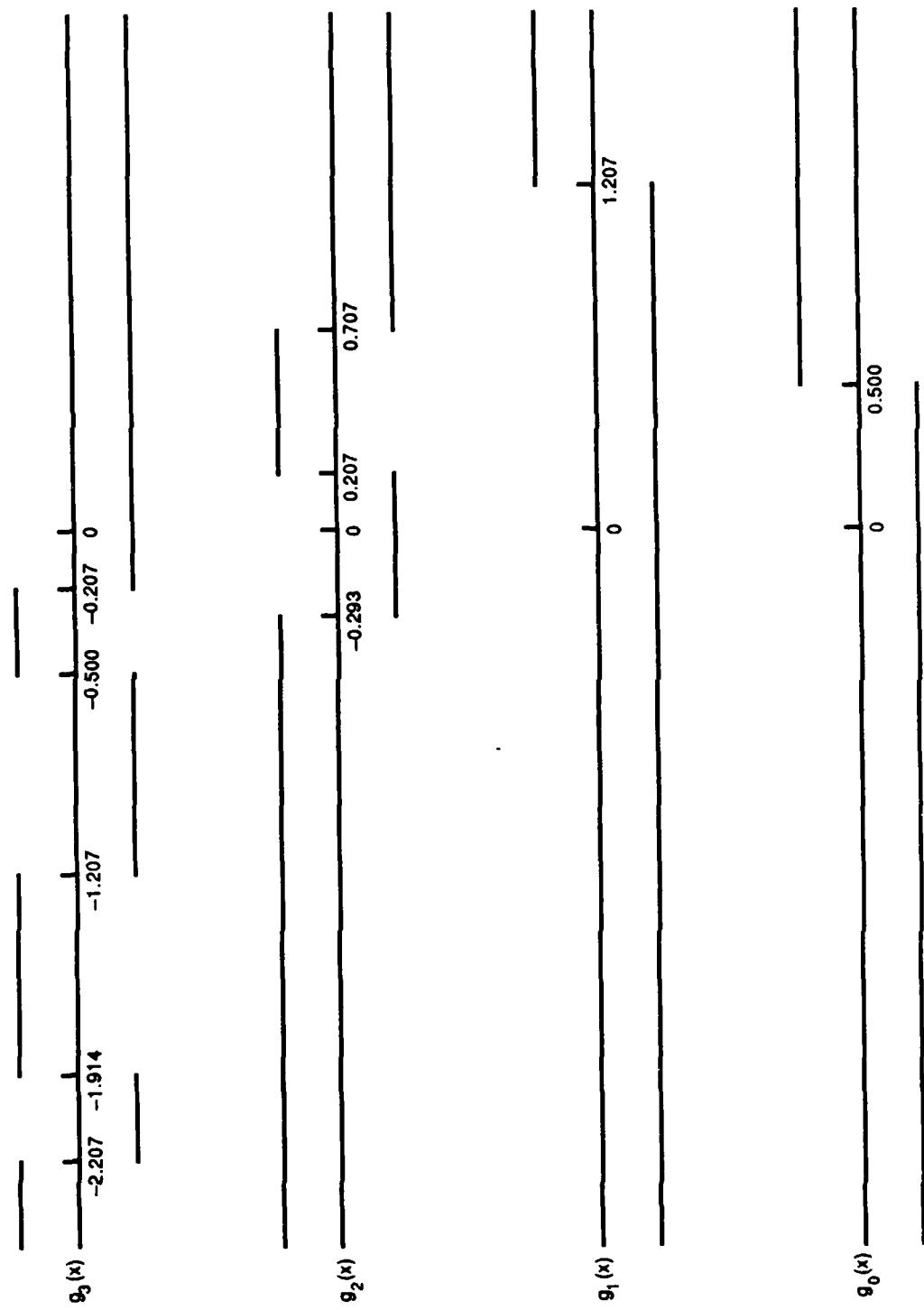


Figure 8. $Z[\sqrt{2}]_{\{-2,-1,0,1\}}$ Converter Functions

3.3 COMPLEX ALGEBRAIC-INTEGER QUANTIZATION

Complex algebraic-integer quantizers correspond to two-dimensional vector quantizers in \mathbf{R}^2 . As such, the encoding problem becomes much more difficult. One obvious approach is to quantize the real and imaginary parts of the complex number separately, combining these representations to form the complex algebraic-integer representative. This approach will be illustrated for the ring $\mathbf{Z}[e^{2\pi i/16}]$, the real numbers of which form the ring $\mathbf{Z}[\sqrt{2 + \sqrt{2}}]$. Let $\omega = e^{2\pi i/16}$.

Suppose $z = x + yi$ is a complex number to be approximated. First, approximations for the real numbers x and y in $\mathbf{Z}[\sqrt{2 + \sqrt{2}}]_M$ (represented using the nonpolynomial basis) are computed:

$$\hat{x} = a_0 + a_1 \sqrt{2 + \sqrt{2}} + a_2 \sqrt{2} + a_3 \sqrt{2 - \sqrt{2}}, \quad (3.9)$$

$$\hat{y} = b_0 + b_1 \sqrt{2 + \sqrt{2}} + b_2 \sqrt{2} + b_3 \sqrt{2 - \sqrt{2}}. \quad (3.10)$$

Then $\hat{z} \equiv \hat{x} + \hat{y}i$ is an approximation of z in $\mathbf{Z}[\omega]$.

However, \hat{z} is represented in terms of the basis

$$\mathbf{B}_2 = \{1, \sqrt{2 + \sqrt{2}}, \sqrt{2}, \sqrt{2 - \sqrt{2}}, i, \sqrt{2 + \sqrt{2}}i, \sqrt{2}i, \sqrt{2 - \sqrt{2}}i\} \quad (3.11)$$

instead of the polynomial basis $\mathbf{B}_1 = \{1, \omega, \dots, \omega^7\}$. The change of basis matrix from \mathbf{B}_2 to \mathbf{B}_1 is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (3.12)$$

Thus, when expressed in terms of the polynomial basis, \hat{z} is in $\mathbf{Z}[\omega]_{2M}$.

The inverse of (3.12) is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}. \quad (3.13)$$

The fractions in the second matrix indicate that \mathbf{B}_2 does not generate all of $\mathbb{Z}[\omega]$; \mathbf{B}_2 is not an integral basis of $\mathbb{Z}[\omega]$. In fact, if $\mathbb{Z}[\mathbf{B}_2]_M$ denotes the points that are represented in terms of the elements of \mathbf{B}_2 with coefficients in the range $[-M/2, M/2]$, then, as a subset of \mathbb{R}^2 , $\mathbb{Z}[\mathbf{B}_2]_M$ is the *cross product*

$$\begin{aligned}\mathbb{Z}[\mathbf{B}_2]_M &= \mathbb{Z}[\sqrt{2 + \sqrt{2}}]_M \times \mathbb{Z}[\sqrt{2 + \sqrt{2}}]_M \\ &= \{(a, b); a, b \in \mathbb{Z}[\sqrt{2 + \sqrt{2}}]_M\},\end{aligned}\quad (3.14)$$

and this rectangular cross product is a proper subset of $\mathbb{Z}[\omega]_{2M}$.

No rigorous study was made of the decrease in performance incurred when real and imaginary parts are approximated separately by the above technique. It is reasonable to expect, however, that the error performance of $\mathbb{Z}[\omega]_M$ in \mathbb{R}^2 is approximated by the error performance of $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]_M$ in \mathbb{R} . (Indeed, [6] showed that this is the case for $\mathbb{Z}[e^{2\pi i/8}]$ and $\mathbb{Z}[\sqrt{2}]$ for a region of \mathbb{R}^2 near the origin.) Thus, it is expected that the error performance for $\mathbb{Z}[\omega]_M$ should be roughly comparable to $\mathbb{Z}[\mathbf{B}_2]_M$. But the representatives obtained lie in $\mathbb{Z}[\omega]_{2M}$, so the cost of the increase in efficiency is, if these assumptions are correct, just the doubling of the dynamic range of the inputs.

If doubling the dynamic range of the inputs has to be avoided, then approximations relative to the basis \mathbf{B}_3 :

$$\left\{1, \sqrt{2 + \sqrt{2}}/2, \sqrt{2}/2, \sqrt{2 - \sqrt{2}}/2, i, \sqrt{2 + \sqrt{2}}i/2, \sqrt{2}i/2, \sqrt{2 - \sqrt{2}}i/2\right\} \quad (3.15)$$

can be computed (the roles of the change of basis matrices (3.12) and (3.13) are interchanged). However, now, $\mathbb{Z}[\mathbf{B}_3]$ contains $\mathbb{Z}[\omega]$ and a strategy must be followed to exclude elements. This is the approach of [14], which was applied in the case $\mathbb{Z}[e^{2\pi i/8}]$, but is equally valid here.

3.4 QUANTIZATION PERFORMANCE FOR $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$

The performance of a quantizer is usually evaluated in terms of signal-to-noise ratios. More precisely, the analog signal x is assumed to be a zero-mean random variable with probability density function p_x and variance σ_x^2 . If \hat{x} represents the quantized version of x , then the mean of the squared error, or *quantization-noise* variance is given by

$$\sigma_q^2 = \int_{-\infty}^{\infty} (x - \hat{x})^2 p(t) dt. \quad (3.16)$$

The *signal-to-quantization-noise ratio (SNR)* is defined as $10 \log_{10}(\sigma_x^2 / \sigma_q^2)$ dB.

This section computes the SNR for quantizers derived from $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$ using the nonpolynomial basis $\{1, \sqrt{2 + \sqrt{2}}, \sqrt{2}, \sqrt{2 - \sqrt{2}}\}$. This basis results in lower dynamic-range growth and, if desired, leads more naturally to complex representatives. Both direct and two-stage algebraic-integer quantizers are considered and compared. The two-stage quantizer is assumed to employ a 12-bit uniform quantizer as the front end.

3.4.1 Uniform Inputs

In this section, the analog input x is assumed to be uniformly distributed on the interval $[-1, 1]$; the signal variance is $\sigma_x^2 = 1/3$. For comparison, the optimal uniform quantizer, which has decision thresholds extending from -1 to 1 , has an SNR of 6.02 dB/bit [10]. The algebraic-integer quantizers are determined by specifying the coefficient range M . The quantizers obtained extend beyond the interval $[-1, 1]$ and so must be scaled (in this case by a fractional amount) to best match the uniform input distribution. A similar scaling (by an easily calculated amount) would be required for the uniform quantizer if it had been initially matched to a larger variance. In the algebraic-integer case, the proper scale factor is determined by computer search.

Figures 9 and 10 show the SNR plotted as a function of \log_{10} of the scale factor for the direct and two-stage quantization schemes, respectively. The scale factor is plotted using the logarithm to facilitate relative error comparisons for nonoptimal choices of scale factor. These plots were obtained using a computer; the value of (3.16) can be evaluated in closed form in this case.

Figures 9 and 10 show that, for both methods, the SNR rises rapidly as the scale factor is increased from a very small value (actually the initial value of the scale factor is .010), reaches a peak at about the same value of the scale factor (actually at the same value), and then gradually declines as the scale factor is increased further (the largest value of the scale factor considered is .550). The gradual fall-off of the curves means that the quantizers perform near optimally over a wide range of scale factors.

The limiting effect of the 12-bit front end in the two-stage method is evident in figure 10. The curves are approaching a ceiling of 72.24 dB—the accuracy of the 12-bit uniform quantizer. However, for the smaller values of M , the curves appear almost to be identical. Table 3 gives the optimal values of the SNR and the corresponding value of the scale factor. For $M \leq 8$ (equivalently, for up to about 11 bits of accuracy), the methods perform similarly. Thus, the following heuristic is indicated: the suboptimal two-stage method performs practically equivalent to the optimal direct method if the front-end A/D converter in the two-stage method is chosen with one bit, or more conservatively with two bits, of accuracy beyond the accuracy requirements of the processor.

The performance of the quantizers derived from the polynomial basis is within about 1 dB of the values in table 3. The scale factors, however, are smaller, reflecting

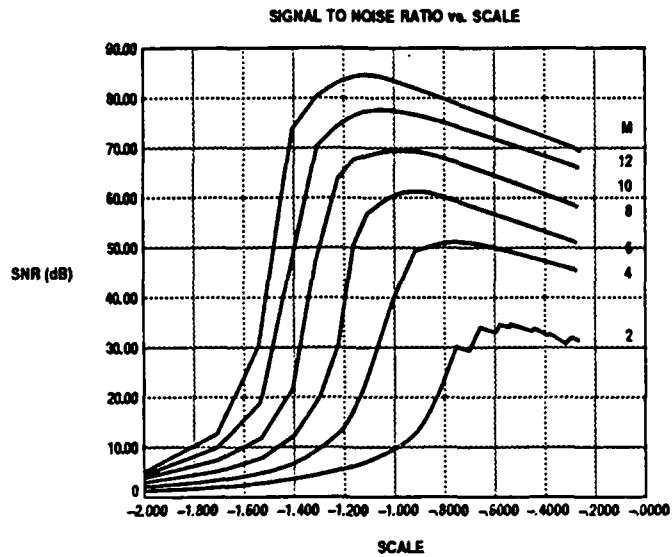


Figure 9. Uniform Inputs; $Z[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$;
Nonpolynomial Basis; Direct Quantizer

the fact that $Z[\sqrt{2 + \sqrt{2}}]_M$ expressed in terms of the polynomial basis involves larger numbers.

It is useful to compare the efficiency of the algebraic-integer quantizers with the more conventional uniform quantizers. The *rate* of a quantizer containing P points is defined as $\log_2 P$. With this definition, ordinary b -bit quantizers have rate b . For any quantizer Q , the *efficiency relative to the uniform quantizer* is defined as the ratio

$$\begin{aligned}
 & \frac{\text{equivalent bit performance of } Q}{\text{rate of } Q} \\
 & = \frac{(\text{performance of } Q \text{ (dB)})/6.02}{\text{rate of } Q}. \tag{3.17}
 \end{aligned}$$

Table 4 lists the efficiencies of the direct algebraic-integer quantizers. Efficiencies of about 90 percent are obtained with slightly increased efficiency for larger values of M .

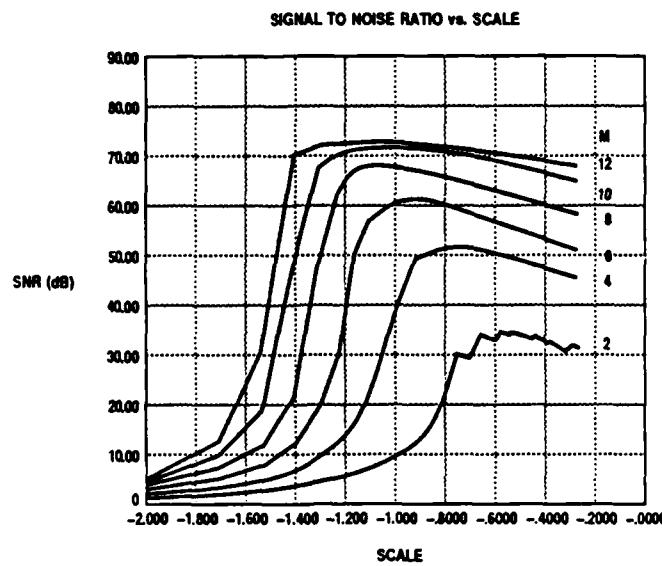


Figure 10. Uniform Inputs; $Z[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$;
Nonpolynomial Basis; Two-Stage Quantizer

Table 3. Comparison of Direct and Two-Stage $Z[\sqrt{2 + \sqrt{2}}]_M$ Quantizers, $2 \leq M \leq 12$; Nonpolynomial Basis; Uniform Inputs

M	Scale	SNR (dB)	
		Direct	Two-Stage
2	.270	34.06	34.06
4	.170	51.35	51.31
6	.120	61.09	60.76
8	.100	68.94	67.25
10	.090	77.43	71.09
12	.080	83.45	71.93

**Table 4. Efficiency of $Z[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$;
Uniform Inputs; Direct Quantizer**

M	Bit Perf.	Rate	Efficiency
2	5.66	6.34	.892
4	8.53	9.29	.918
6	10.09	11.23	.904
8	11.45	12.68	.903
10	12.86	13.84	.930
12	13.86	14.80	.937

The fractional efficiencies are, of course, consistent with the fact that, if the inputs are uniformly distributed, the uniform quantizer performs the best.

The algebraic integers form a *distributed* representation. For example, if 8-bit performance is desired, then, examining table 4, representations with coefficient range $M = 4$ can be used. This quantizer yields 8.53 bits of accuracy with a number of points that corresponds to 9.29 bits. The conventional 8-bit quantizer involves a single value, ranging from -128 to 127 for 2's-complement representation. The algebraic-integer quantizer involves four values, each ranging from -2 to 2.

In actual systems, the variance of the input signal is often not known exactly, or is changing with time (assuming there is even a fixed probability distribution). In such situations it, is important that a quantizer perform well over a wide range of possible input variances. To compare the conventional and algebraic-integer quantizers on this point, the efficiency of the direct algebraic-integer quantizers is calculated for the case that the input is assumed to be uniformly distributed on $[-1, 1]$, but in fact the inputs are uniformly distributed on $[-1/2, 1/2]$ (a mismatch in variance by a factor of $1/4$).

The performance for the uniform quantizer in this case degrades by exactly one bit, or 6.02 dB. A b -bit uniform quantizer on $[-1, 1]$ with inputs from $[-1/2, 1/2]$ performs like a $(b - 1)$ -bit quantizer. The performance of the algebraic-integer quantizer is given by the SNR in figure 9 at a value of twice the optimal scale factor s_0 (or $s_0 + .301$ on the log scale). These values for $M = 2, 4, \dots, 12$ are, respectively, 31.10, 48.69, 57.52, 65.82, 74.52 and 79.51. Note that the degradation from the optimal algebraic-integer performance given in table 3 averages around 3.2 dB, resulting in an increased efficiency for this variance mismatch case. Table 5 lists these efficiencies, which now average around 96 percent.

Table 5. Efficiency of $Z[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$; Uniform Inputs; Direct Quantizer; Variance Mismatch of 1/4

M	Bit Perf.	Rate	Efficiency
2	6.16	6.34	.972
4	9.09	9.29	.978
6	10.55	11.23	.939
8	11.93	12.68	.941
10	13.38	13.84	.967
12	14.21	14.80	.960

3.4.2 Gaussian Inputs

To test the performance of the algebraic-integer quantizers when the input is nonuniformly distributed, this section assumes the analog input x is Gaussian (or normally) distributed. Other common nonuniform input distributions, especially for speech and images, are Laplacian and Gamma distributions, but they will not be considered here. So that the input power in this and the last section are identical, the variance is taken as $\sigma_x^2 = 1/3$; the mean is still assumed to be zero. For reference, table 6 lists the SNR for the optimum uniform quantizers with 1 through 16 bits. The values in table 6 for 1 through 8 bits were obtained from [10], page 127; the values for 9 through 16 bits were estimated—the step size used was extrapolated from the step sizes for 1 through 8 bits obtained from [10].

Again, the algebraic-integer quantizer must be scaled to best match the Gaussian input distribution. Figures 11 and 12 show the SNR plotted as a function of \log_{10} of the scale factor for the direct and two-stage quantization schemes, respectively. These plots were obtained using a computer; the value of (3.16) in this case must be evaluated numerically.

Figures 11 and 12 show that, for both methods, the SNR rises as the scale factor is increased from a very small value (actually the initial value of the scale factor is .010), reaches a peak at about the same value of the scale factor, and then declines more gradually as the scale factor is increased further (the largest value of the scale factor considered is .562). The gradual fall-off of the curves means that the quantizers perform near optimally over a wide range of scale factors.

The limiting effect of the 12-bit front end in the two-stage method is evident in figure 12. The curves are approaching a ceiling of 62.70 dB—the accuracy of the 12-bit uniform quantizer for a Gaussian input. However, for the smaller values of M , the

**Table 6. SNR of Optimal Uniform Quantizer;
Gaussian Input**

Bits	SNR (dB)
1	4.40
2	9.25
3	14.27
4	19.38
5	24.57
6	29.83
7	35.13
8	40.34
9	46.03
10	51.54
11	57.10
12	62.70
13	68.33
14	74.00
15	79.68
16	85.39

curves appear almost to be identical. Table 7 gives the optimal values of the SNR and the corresponding value of the scale factor for the direct case (the scale factor for the two-stage approach is within .03 of this value). In this case, there is a slight performance degradation almost immediately, which becomes significant by the time $M = 8$. However, up to $M = 6$, which from examination of table 6 corresponds to better than 11 bits of accuracy, the performances are more comparable. This confirms the heuristic derived in the last section: the suboptimal two-stage method performs practically equivalent to the optimal direct method if the front-end A/D converter in the two-stage method is chosen with one bit, or more conservatively with two bits, of accuracy beyond the accuracy requirements of the processor.

It is more difficult in this case to determine the efficiency of the algebraic-integer quantizers relative to the more conventional uniform quantizers. This is because there is no simple formula, such as $(6.02)(\text{number of bits})$, for the performance of the optimal uniform quantizer against a Gaussian input. The equivalent bit performance for the algebraic-integer quantizer is estimated by linearly interpolating between the performances of two successive bit values that contain the performance of the algebraic-integer quantizer. Table 8 lists the efficiencies of the direct algebraic-integer quantizers. Effi-

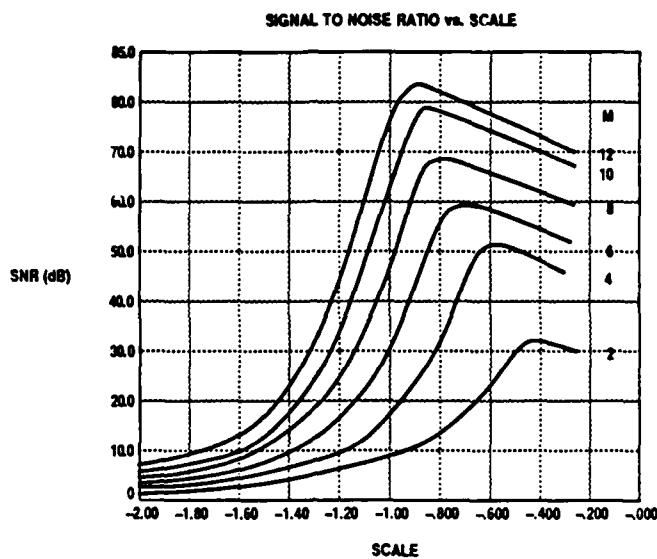


Figure 11. Gaussian Inputs; $Z[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$;
Nonpolynomial Basis; Direct Quantizer

ciencies greater than 1 are consistent with the fact that the nonuniformly distributed algebraic-integer quantizers better match the Gaussian input distribution.

For example, examining table 8, representations with coefficient range $M = 4$ yield 9.82 bits of accuracy with a number of points that corresponds to 9.29 bits. These 9.29 bits are distributed among four coefficients, each ranging from -2 to 2 .

3.5 CONCLUSION

Analog-to-algebraic-integer conversion was considered. The focus of the section was on real algebraic integers, since the complex case is further complicated by the need to perform vector quantization in two dimensions. It was shown how separate scalar quantization of the I and Q channels could be combined to form a complex algebraic-integer representative, expressed in terms of a product basis. Other, more desirable, basis representations are obtained by transforming coefficients, resulting in represen-

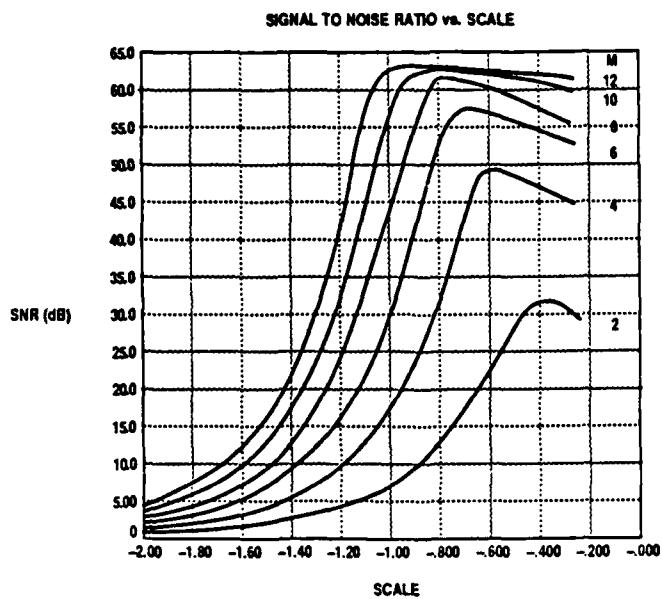


Figure 12. Gaussian Inputs; $Z[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$;
Nonpolynomial Basis; Two-Stage Quantizer

Table 7. Comparison of Direct and Two-Stage $Z[\sqrt{2 + \sqrt{2}}]_M$
Quantizers, $2 \leq M \leq 12$; Nonpolynomial Basis; Gaussian Inputs

M	Scale	SNR (dB)	
		Direct	Two-Stage
2	.3981	31.60	31.11
4	.2512	50.56	49.21
6	.1995	59.47	57.02
8	.1585	65.87	61.18
10	.1413	78.43	62.35
12	.1122	82.21	62.50

**Table 8. Efficiency of $Z[\sqrt{2 + \sqrt{2}}]_M$, $2 \leq M \leq 12$;
Gaussian Inputs; Direct Quantizer**

M	Bit Perf.	Rate	Efficiency
2	6.33	6.34	.998
4	9.82	9.29	1.057
6	11.42	11.23	1.017
8	12.56	12.68	.991
10	14.78	13.84	1.068
12	15.44	14.80	1.043

tatives with accuracies that are *conjectured* to be equivalent to accuracies obtainable from the two-dimensional vector quantizers, but at a cost of twice the coefficient range.

Two methods that perform direct real algebraic-integer conversion were described. The first method required the implementation of the exact shape of a nonlinear function. The second method also involved nonlinear functions, but only required that the zero crossings of these functions be implemented accurately. However the complexity of the second method grows quickly.

A more practical two-stage approach, which is based on conventional A/D converter technology was introduced. Although this approach is in theory suboptimal, it is practically equivalent (at least in the case considered: $Z[\sqrt{2 + \sqrt{2}}]$) to the optimal direct approach, provided that the processor is operated at one to two bits below the accuracy of the front-end A/D converter. A corollary of this observation is that further hardware development of a direct analog-to-algebraic-integer converter is not warranted unless the application requires processing at accuracies and speeds that are at the limits of conventional A/D technology.

Finally, the algebraic integers form a distributed quantizer—a single large representative is replaced with a representative consisting of a vector of smaller coefficients. For the same level of accuracy, the total cost (measured by the number of points) of the conventional and algebraic-integer quantizers is comparable. For uniformly-distributed inputs, the algebraic-integer quantizers (based at least on the case considered: $Z[\sqrt{2 + \sqrt{2}}]$) averaged about 91 percent as efficient as the conventional quantizers, although in practice the robustness of the algebraic-integer quantizers would tend to increase this efficiency. For Gaussian-distributed inputs, the algebraic-integer quantizers averaged about 103 percent as efficient. An advantage of the distributed representation is that the smaller coefficients reduce the complexity of the integer-RNS conversion into and out of the outer level of parallelism, which is considered next.

SECTION 4

INTEGER RNS CONVERSION: THE OUTER LEVEL OF PARALLELISM

In an algebraic-integer RNS implementation, integer RNS is applied separately to each coefficient of the algebraic integer to form the outer level of parallelism. Converting into the outer level of parallelism is expected to be straightforward because the distributed algebraic-integer representatives have small coefficients, which are likely to be smaller than the moduli in the RNS. Converting out of the outer level of parallelism will be more complicated, although the complexity will be mitigated somewhat by the smaller RNS ranges, resulting from smaller input ranges, that are needed to contain the dynamic range of individual coefficients. As is the case for integer RNS, this conversion phase could still very well be the bottleneck in an algebraic-integer RNS implementation.

In an attempt to eliminate this bottleneck, most converters have either taken advantage of specialized sets of moduli (e.g., the set $\{2^n - 1, 2^n, 2^n + 1\}$ used in [16] and [21]), or have used mixed-radix conversion. Unfortunately, the first approach is severely limited by the availability of appropriate sets of moduli for algebraic-integer implementations, where the moduli have to satisfy certain restrictions to guarantee the further splitting into the inner channels. The second approach, mixed-radix conversion, is convenient for many applications, but suffers limitations of its own. One of these limitations is its lack of flexibility in being reprogrammed for different moduli as it requires $\mathcal{O}(n^2)$ modulo adders, which are not easily reconfigurable, for an n -modulus RNS conversion.

This section looks at a recently proposed method for rapid output conversion that overcomes some of these difficulties, called *fractional representation*. The next section explicates conversion using fractional representation, and modifies it slightly by replacing lookup tables with binary multipliers. Subsequently, a generalization allowing adjustable output precision is developed and a comparison is made with mixed-radix conversion. It will be shown that fractional-representation conversion is fast, requires no custom modulo circuitry, has adjustable output precision to match the system precision, and is adaptable to a variety of moduli combinations even after being designed and laid down on silicon.

4.1 FULL-PRECISION OUTPUT CONVERSION

Given an RNS $\{m_1, m_2, \dots, m_n\}$, $M = \prod m_i$, the Chinese Remainder Theorem guarantees that any integer X between 0 and $M - 1$, inclusive, can be reconstructed from its residues x_i . This is done by using the formula ([15], pp. 30-31),

$$X = \left\langle \sum_{i=1}^n \frac{M}{m_i} \langle w_i x_i \rangle_{m_i} \right\rangle_M, \quad (4.1)$$

where the w_i satisfy the equation,

$$\left\langle \frac{M}{m_i} w_i \right\rangle_{m_i} = 1.$$

Constructing an output converter via a straightforward application of (4.1) is problematic, however, as the large reduction modulo M at the end requires custom circuitry, which is more costly to operate than an ordinary binary adder (figure 13).

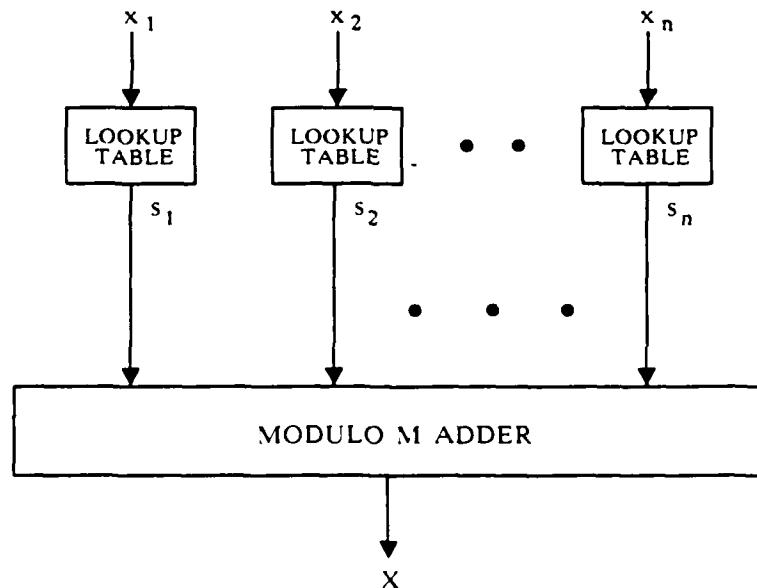


Figure 13. Residue Decoding by the Chinese Remainder Theorem

4.1.1 Fractional Representation

An equivalent way to write equation (4.1) is¹,

$$X = \sum_i \frac{M}{m_i} \langle w_i x_i \rangle_{m_i} - MK, \quad (4.2)$$

where K is some nonnegative integer. The difficulty in using (4.2) lies in reducing the sum by a multiple of M to lie in the range $[0, M - 1]$, when numbers are represented in binary. This can be avoided by scaling X so that the result lies in the range $[0, 2)$. Multiplying X by $2/M$, and calling the result X_s , yields

$$X_s = \sum_i \frac{2}{m_i} \langle w_i x_i \rangle_{m_i} - 2K. \quad (4.3)$$

In this way, the reduction modulo M can be transformed into a simple discarding of bits above the unit bit. The true output X can now be fully recovered by multiplying X_s by $M/2$, though in many cases a scaled version of the output X will suffice. This is an important point, which will be expanded upon in section 4.2.

The remaining consideration in computing X_s is the representation of the summand u_i in (4.3), $u_i = \frac{2}{m_i} \langle w_i x_i \rangle_{m_i}$. Notice that $0 \leq u_i < 2$ and the fractional part of u_i cannot be represented exactly in binary unless m_i is a power of 2. Let u_i be approximated by \hat{u}_i , where

$$\hat{u}_i = \lceil 2^t u_i \rceil 2^{-t}, \quad (4.4)$$

and the ceiling brackets indicate the closest integer greater than or equal to the enclosed quantity (the following analysis could be easily modified to accommodate rounding or truncation to arrive at the approximations). Now, $t + 1$ bits are used to approximate u_i , with 1 bit for the integer part and t bits for the fractional part. From equation (4.4), it can be seen that there is some approximation error e_i such that $\hat{u}_i = u_i + e_i$, where e_i satisfies the inequality² $0 \leq e_i < 2^{-t}$. Using \hat{u}_i instead of u_i in (4.3) introduces an error e into the computed value of X_s ,

$$e = \sum_i^n e_i < n2^{-t}. \quad (4.5)$$

¹Except where noted, the following material up to (4.6) is condensed from [22].

²The authors in [18] and [22] state that, once the RNS is fixed, the upper bound on e_i can be made exact by an exhaustive search of all m_i possible fractions. Actually, the exact upper bound can be shown to be

$$e_i \leq \left[1 - 2^{\min\{q_i, t+1\}} / m_i \right] 2^{-t}$$

where q_i is the number of factors of 2 contained in m_i .

Since the correct possible values of X_s are evenly spaced by increments of $2/M$, exact reconstruction is possible, by rounding down to the next correct value, if the error satisfies the inequality $e < 2/M$. By the bound above on e , the inequality is satisfied when $n2^{-t} \leq 2/M$, or equivalently

$$t \geq \lceil \log_2 Mn \rceil - 1 \quad (4.6)$$

(the ceiling brackets are included to guarantee that t , the number of bits representing the fractional part of \hat{u}_i , is an integer as required in any practical system). This means that if \hat{u}_i is calculated by a lookup table accepting x_i as input and outputting \hat{u}_i , that lookup table must store $t + 1 = \lceil \log_2 Mn \rceil$ bits for each possible value of x_i .

4.1.1.1 Numerical Example

To illustrate how this procedure works, take the RNS consisting of the three largest 5-bit representable primes — 23, 29, and 31. The range $M = 20,677$ and the weights w_i are 12, 12, and 2, respectively. Suppose $X = 9921$. Its residues are $x_1 = 8$, $x_2 = 3$, and $x_3 = 1$. There are three terms in the sum in (4.3),

$$\sum_{i=1}^3 u_i = \frac{2}{23} (12 \times 8)_{23} + \frac{2}{29} (12 \times 3)_{29} + \frac{2}{31} (2 \times 1)_{31},$$

which are used to calculate X_s . As indicated in (4.6), the u_i s are approximated by 16-bit numbers:

$$\begin{aligned} \hat{u}_1 &= \left[2^{15} \cdot \frac{8}{23} \right] 2^{-15} = 0.010110010000110, \\ \hat{u}_2 &= \left[2^{15} \cdot \frac{14}{29} \right] 2^{-15} = 0.011110111001100, \\ \hat{u}_3 &= \left[2^{15} \cdot \frac{4}{31} \right] 2^{-15} = 0.001000010000101. \end{aligned}$$

Given the residues as input, the above numbers are fetched from the lookup table and added:

$$\begin{array}{r} \hat{u}_1 : 0.010110010000110 \\ \hat{u}_2 : 0.011110111001100 \\ + \hat{u}_3 : 0.001000010000101 \\ \hline \hat{X}_s : 0.111101011010111 \end{array}$$

Converting to decimal, $\hat{X}_s = 0.959686\cdots$. Multiplying by $M/2$ gives the result 9921.72..., which is truncated to the value $\hat{X} = 9921$.

Notice that if only 15 bits had been used for the approximation, the final answer would have been incorrect. In that case, the sum would have been

$$\begin{array}{r} 0.01011001000011 \\ 0.01111011100110 \\ + 0.00100001000011 \\ \hline 0.11110101101100 \end{array}$$

or $\hat{X}_s = 0.959716\cdots$, yielding a final answer of 9922.

4.1.2 Modified Fractional Representation

X_s may also be calculated in a way that avoids lookup tables or RAMs altogether. This is due to the fact that the inner reduction modulo m_i in (4.1) need not be performed, i.e.,

$$X = \left\langle \sum_i \frac{M}{m_i} w_i x_i \right\rangle_M. \quad (4.7)$$

This equation may be transformed, via the same process used previously, into

$$X_s = \sum_i^n \frac{2}{m_i} w_i x_i - 2L, \quad (4.8)$$

where L is some nonnegative integer. It can be seen that the summand may now be computed simply by multiplying $\frac{2}{m_i} w_i$ by the residue x_i . Unfortunately, the first multiplicand cannot be represented exactly in binary, unless the denominator in the fraction when reduced to its simplest form is a power of 2. Call the first multiplicand y_i , and let \hat{y}_i be an $(s+1)$ -bit approximation of y_i such that

$$\hat{y}_i = \lceil 2^s y_i \rceil 2^{-s}. \quad (4.9)$$

As $0 \leq y_i < 2$, the first bit represents the integer part of \hat{y}_i with the remaining s bits representing the fractional part. The approximation of y_i causes an additive error f_i to be introduced into \hat{y}_i , where³ $0 \leq f_i < 2^{-s}$. Using $\hat{v}_i = \hat{y}_i x_i$ instead of u_i in computing X_s causes a cumulative error f ,

$$f = \sum_i^n f_i x_i < 2^{-s} \sum_i^n (m_i - 1), \quad (4.10)$$

³The additive error can be calculated exactly from the equation

$$f_i = \left[\left\langle -w_i \cdot 2^{s+1} \right\rangle_{m_i} / m_i \right] 2^{-s}.$$

to be added to the final result. By the argument presented earlier, exact reconstruction of the output is possible if the error f is less than $2/M$. This is true when

$$2^{-s} \sum_i^n (m_i - 1) \leq 2/M,$$

or equivalently

$$s \geq \left\lceil \log_2 M \sum_i^n (m_i - 1) \right\rceil - 1. \quad (4.11)$$

In sum, the reconversion may be performed by ordinary binary multipliers that multiply each residue x_i by a fixed weight of $s + 1$ bits. As the result has only s bits of fractional accuracy, and as all bits above the unit bit are irrelevant, only the lowest $s + 1$ bits of the product are needed in order to compute X_s .

4.1.2.1 Numerical Example Revisited

To compare the above with the lookup table implementation, consider the example shown previously. In this instance, there are again three terms in the sum,

$$\sum_{i=1}^3 v_i = \left(\frac{2}{23} \cdot 12 \right) 8 + \left(\frac{2}{29} \cdot 12 \right) 3 + \left(\frac{2}{31} \cdot 2 \right) 1,$$

used to calculate X_s . The terms in parentheses are the weights y_i . By (4.11), these weights must be approximated by numbers of at least 21-bit accuracy:

$$\begin{aligned} \hat{y}_1 &= \left\lceil 2^{20} \cdot \frac{24}{23} \right\rceil 2^{-20} = 1.00001011001000010111, \\ \hat{y}_2 &= \left\lceil 2^{20} \cdot \frac{24}{29} \right\rceil 2^{-20} = 0.11010011110111001100, \\ \hat{y}_3 &= \left\lceil 2^{20} \cdot \frac{4}{31} \right\rceil 2^{-20} = 0.00100001000010000101. \end{aligned}$$

The \hat{y}_i are multiplied by the residues $\{8, 3, 1\}$ and the 21 lowest bits of each product are summed to form \hat{X}_s :

$$\begin{aligned} \hat{y}_1 x_1 &: 0.01011001000010111000 \\ \hat{y}_2 x_2 &: 0.01111011100101100100 \\ + \hat{y}_3 x_3 &: 0.00100001000010000101 \\ \hline \hat{X}_s &: 0.1111010110101010100001 \end{aligned}$$

Converting to decimal, $\hat{X}_s = 0.959626\cdots$. Scaling up by $M/2$ yields the value 9921.09 \cdots , which is truncated to the correct result, $\hat{X} = 9921$.

4.2 ADJUSTABLE PRECISION OUTPUT CONVERSION

In the previous section, it was assumed that a full-precision integer was the desired output of the residue-to-binary converter. This, however, is not always the case. Consider for a moment the nature of an integer-RNS system. Usually, it takes some input(s) upon which it performs additions and multiplications. The range that the resulting output can lie in is almost certainly larger than the input size, and in most cases is considerably larger.

To illustrate this claim, suppose an FIR filter (a situation in which RNS is often used) is being given a stream of 8-bit inputs. Suppose in addition that there are 128 filter coefficients, each with 8-bit magnitude. In such a case, the outputs could be as large as 23 bits. However, the number of *significant* bits could range anywhere from 16 bits in the case of a matched filter, to 8 bits for an ordinary bandpass filter, to less than that. Even in the worst case 7 bits of output precision would be unnecessary and could be discarded.

There is some reason to believe that higher conversion accuracy may be required for algebraic-integer RNS applications, but it is not necessarily the case that *full* output precision will always be required even then (for example, see sections 6.1 and 6.2). The reader should be aware that the amount of output conversion precision is a more sensitive issue when an algebraic-integer RNS, as opposed to an ordinary integer RNS, is employed.

It is clear from the preceding rough calculation that considerable savings might result if there were some way to take advantage of the less-than-full output precision necessary in many situations. For mixed-radix conversion, it is known that the large adders at the end can be reduced if less than the full output precision is desired, but that this in no way reduces the amount of computation required for the mixed-radix coefficients. In the following subsections it will be shown that the hardware required in fractional-representation conversion scales almost linearly with the number of output bits desired.

4.2.1 Fractional Representation

Suppose it is decided that the lowest g bits of the output from the residue-to-binary converter are unnecessary. That is, instead of resolving to the nearest $2/M$ increment, it is decided that the output should be resolved only to the nearest increment of $2^{g+1}/M$.

This implies that the conversion error e should be less than the desired resolution, $2^{g+1}/M$.

Let \hat{u}_i be a $(t + 1)$ -bit approximation of u_i , as before. Then, by the bound on e developed in equation (4.5), the conversion error inequality is satisfied when $n2^{-t} \leq 2^{g+1}/M$. This is equivalent to requiring that the following inequality hold:

$$t \geq \lceil (\log_2 Mn) - g \rceil - 1. \quad (4.12)$$

As the number of bits of output resolution is $(\log_2 M) - g$, it is clear that the number of bits of accuracy that must be carried (and the size of the hardware as well) goes down linearly with output resolution bits as indicated previously, except for a $(\log_2 n) - 1$ offset.

Suppose that instead of discarding the lowest g bits, the highest h bits were thrown away. This could occur, for instance, if it were known that the output didn't use the full range, but only grew as large as some fraction of M . Specifically, deciding that h high-order bits are unnecessary corresponds to knowing that the integer outputs lie in the range $[0, 2^{-h}M)$. Given the above situation, the h high-order bits can be dropped from the approximations. Thus, the number of bits coming out of a lookup table would only have to satisfy the inequality

$$t \geq \lceil (\log_2 Mn) - h \rceil - 1. \quad (4.13)$$

More generally, if it is known that the output X_s will lie in some subrange such that for nonnegative integers k and h , $0 \leq k < 2^h$,

$$k2^{-h+1} \leq X_s < (k + 1)2^{-h+1},$$

then the highest order h bits of the output X_s are unnecessary and may be dispensed with in the approximations of the summands u_i , as indicated in (4.13).

4.2.1.1 Numerical Example

Consider again the numerical example from section 4.1. Suppose that $g = 4$. Then, as indicated in (4.12), the u_i 's should be approximated by 12-bit numbers:

$$\begin{aligned}\hat{u}_1 &= \left\lceil 2^{11} \cdot \frac{8}{23} \right\rceil 2^{-11} = 0.01011001001, \\ \hat{u}_2 &= \left\lceil 2^{11} \cdot \frac{14}{29} \right\rceil 2^{-11} = 0.01111011101, \\ \hat{u}_3 &= \left\lceil 2^{11} \cdot \frac{4}{31} \right\rceil 2^{-11} = 0.00100001001.\end{aligned}$$

Adding these numbers,

$$\begin{array}{r} \hat{u}_1 : 0.01011001001 \\ \hat{u}_2 : 0.01111011101 \\ + \hat{u}_3 : 0.00100001001 \\ \hline \hat{X}_s : 0.11110101111 \end{array}$$

and converting to decimal yields $\hat{X}_s = 0.960449\dots$. Multiplying by $M/2$ gives $9929.60\dots$, which is truncated to the nearest increment of $2^g = 16$, $\hat{X} = 9920$. In practice, the last step of generating \hat{X} from \hat{X}_s would not be done. Instead, \hat{X}_s would simply be truncated to the number of output bits desired (presumably $\log_2 M - g$ bits) and the result would be the converter output.

For this example, $t + 1$ was 16 bits for a full-precision output. This was reduced to 12 bits by decreasing the converter resolution. Since the hardware scales linearly with $t + 1$, this implies a 25% reduction in hardware. This compares to a 28% reduction ($\log_2 M$ vs. $\log_2 M - g$) in the number of bits of resolution.

4.2.2 Modified Fractional Representation

The results of section 4.2.1 may be straightforwardly extended to cover modified fractional representation as well. Suppose it is desired that the output X_s be resolved to the nearest increment of $2^{g+1}/M$. Then the conversion error f must be less than that increment. Using the bound on f from equation (4.10), it can be seen that this occurs when

$$2^{-s} \sum_i^n (m_i - 1) \leq 2^{g+1}/M,$$

which is equivalent to

$$s \geq \left\lceil \left(\log_2 M \sum_i^n (m_i - 1) \right) - g \right\rceil - 1. \quad (4.14)$$

In this case, except for the offset $\log_2 \sum_i^n (m_i - 1)$, the hardware again scales linearly with the fraction of output precision desired.

In the same fashion as in section 4.2.1, bits can be dropped from the high end of the approximations in order to focus in on smaller subranges of the output range. A subrange of size h bits smaller than the full range size allows the number of bits needed to approximate y_i to be shrunk such that

$$s \geq \left\lceil \left(\log_2 M \sum_i^n (m_i - 1) \right) - h \right\rceil - 1. \quad (4.15)$$

As before, only the least significant $s + 1$ bits of the product $\hat{y}_i x_i$ need be computed to calculate \hat{X}_s .

4.2.2.1 Numerical Example Revisited

Suppose $g = 4$, as previously. Then the weights y_i must be approximated by numbers of 17-bit accuracy, as indicated in (4.14):

$$\begin{aligned}\hat{y}_1 &= \left\lceil 2^{16} \cdot \frac{24}{23} \right\rceil 2^{-16} = 1.0000101100100010, \\ \hat{y}_2 &= \left\lceil 2^{16} \cdot \frac{24}{29} \right\rceil 2^{-16} = 0.1101001111011101, \\ \hat{y}_3 &= \left\lceil 2^{16} \cdot \frac{4}{31} \right\rceil 2^{-16} = 0.0010000100001001.\end{aligned}$$

Multiplying the \hat{y}_i s by the residues $\{8, 3, 1\}$, the lowest 17 bits of each product are summed to form \hat{X}_s :

$$\begin{aligned}\hat{y}_1 x_1 &: 0.0101100100010000 \\ \hat{y}_2 x_2 &: 0.0111101110010111 \\ + \hat{y}_3 x_3 &: 0.0010000100001001 \\ \hline \hat{X}_s &: 0.1111010110110000\end{aligned}$$

Converting to decimal, $\hat{X}_s = 0.959716\ldots$. Scaling by $M/2$ yields the value $9922.03\ldots$. This is truncated to the nearest increment of 16, or 9920. Again, as in the case of ordinary fractional representation, the $M/2$ scaling would not be done in practice. Rather, the output \hat{X}_s would be truncated to the desired number of output bits.

Reducing $s + 1$ by 4 bits caused a 19% reduction in hardware, as compared to a 28% reduction in output resolution. The relative hardware reduction is smaller for the modified fractional representation, as the offset term in (4.14) is larger than that in (4.12).

4.3 COMPARISON OF CONVERSION METHODS

Mixed-radix conversion has been the method of choice for RNS output conversion in recent years. Therefore, any new conversion method must be compared with it to have any chance of gaining widespread acceptance. Unfortunately, direct comparison of conversion methods is often difficult as the criteria for optimality are highly dependent on the nature of the rest of the RNS system. In addition, such factors as the availability of certain technologies and hardware, the need to be able to reprogram the converter

easily, as well as others, may play an important role in any comparison. This means that absolute, final statements as to which method is better cannot be made. What can be done is to compare certain key features and highlight various trade-offs in order to help the reader judge the suitability of the proposed technique, given a specific situation.

The rest of this section attempts to do such a comparison. The first subsection briefly compares mixed-radix conversion with ordinary fractional representation conversion by looking at the general architectures for both, and by comparing converters designed for a specific RNS using the same technology. The next subsection shows that there are reasons for using modified fractional representation above and beyond mere area considerations.

4.3.1 Mixed Radix vs. Fractional Representation

The main advantages of mixed-radix conversion are that it requires no large modulo M reduction at the end, and that it has a somewhat regular, pipelineable structure. A block diagram of a four-modulus mixed-radix converter can be seen in figure 14. The first observation one could make about the structure shown is that the area seems to divide fairly equally between the computation of the mixed-radix coefficients and the summing of the weighted coefficients at the end. A second observation is that the connectivity is somewhat complex — there are several long wires that cross others on their way to their destinations.

A block diagram of an equivalent fractional-representation converter can be seen in figure 15. It is quite apparent that the wiring in this diagram is extremely simple. There are no long lines and no crossing wires. Another observation is that the maximum length path in the mixed-radix converter is longer than that in the fractional representation converter. In fact, it can be seen that all paths are the same for the converter in figure 15, which implies that it is more efficient in some sense than the mixed-radix converter.

There was much discussion previously that the full output precision of the particular RNS under consideration was not usually needed, and that an almost linear reduction in hardware for the fractional-representation converter could be achieved by taking advantage of this and eliminating needless bits. What are the properties of the mixed-radix converter given the same situation? It can be shown that the adders at the end reduce more than linearly with decreasing output precision bits. Offsetting this quick reduction, however, is the fact that no reduction is possible in the computation of the mixed-radix coefficients. The reason for this is that each coefficient is needed to compute the next highest one, and as the operations involved take place within a finite ring, exact representation of the coefficients is always necessary.

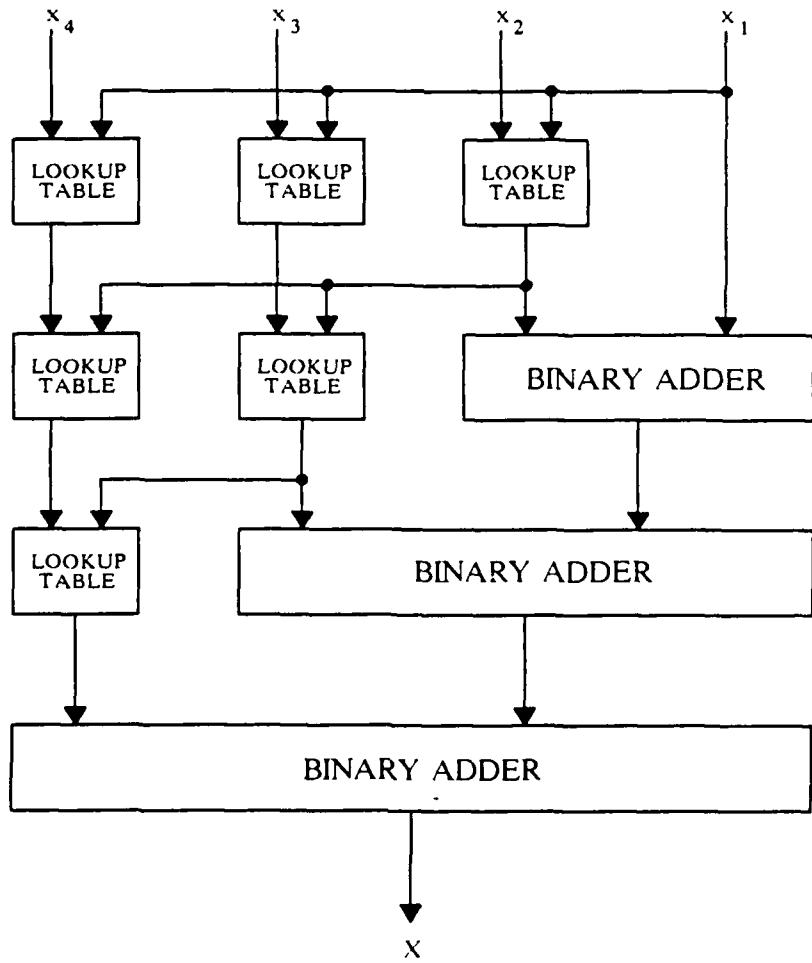


Figure 14. Four-Modulus Mixed-Radix Converter

A better idea of the trade-offs between mixed-radix and fractional-representation conversion in VLSI can be gotten by looking at a specific example. In 1985, Department D-82 of the MITRE Corporation designed a mixed-radix converter for the RNS

$$\mathcal{M} = \{61, 53, 41, 37, 29\}.$$

Based on the technology used in that design, a number of mixed-radix and ordinary fractional-representation converters were designed for the RNS \mathcal{M} with varying degrees of output accuracy. It was found that the active VLSI area of the modified fractional-

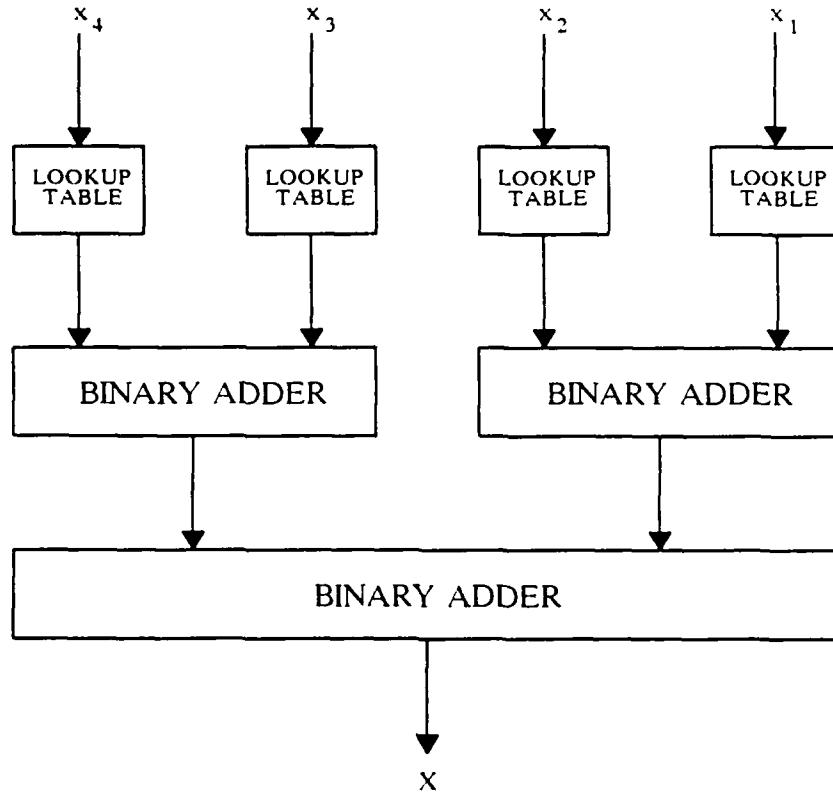


Figure 15. Four-Modulus Fractional-Representation Converter

representation converter as a percentage of the area of the equivalent mixed-radix converter, varied from 90% for full output precision (i.e., 28 bits), to 75% for 12-bit output precision, and approached a limit of 20% as the number of output precision bits went to zero. This area advantage occurred in addition to the regularity advantage of the fractional representation architecture mentioned earlier.

4.3.2 Advantages of Modified Fractional Representation

The full power of the fractional representation is only realized when it is modified to allow ordinary (i.e., no modulo operations) weighting of the residues in the reconstruction formula. Ordinary weighting enables one to use standard binary multipliers rather than modulo-specific lookup tables, thereby freeing the designer from the task of designing a new converter for each and every RNS. Using modified fractional repre-

sentation, a general purpose converter can be built which can be programmed for the RNS desired merely by hardwiring the necessary weighting factors to the inputs of the multipliers.

A VLSI design using the modified fractional representation was beyond the scope of this project. However, other efforts within Department D-82 have yielded very nice results in that direction. A systolic array that performs modified fractional-representation conversion was developed, in which each array cell consists of a small adder with a small amount of controlling circuitry. In addition, an input conversion algorithm was found which maps directly into the same systolic array, yielding a regular, programmable architecture capable of RNS I/O conversion. The array is programmed for a specific RNS in the manner described above. Since each cell in it operates as fast, if not faster than, the inner RNS operations, the systolic array converter imposes no constraints on the throughput of the overall RNS system. A paper documenting these results is in preparation.

4.4 CONCLUSION

A new method of RNS output conversion, fractional representation, has been explained and a slightly modified version of it employing binary multipliers rather than lookup tables has been derived. The new method is fast, flexible, and yields a fairly simple structure, which should be easily realizable in hardware. A general architecture comparison, as well as a VLSI area comparison for a specific example, was made between mixed-radix and ordinary fractional-representation converters. In the VLSI area comparison, the area of the fractional-representation converter was always less than that of the equivalent mixed-radix converter. The main advantage of fractional representation occurs in the modified version, which allows a general output converter to be designed. Such a design was pursued under a separate effort. A systolic array using modified fractional representation has been developed which can also perform RNS input conversion. The array is programmable for any RNS desired (within certain designer-chosen limits) and operates at least as fast as the inner RNS operations that it is meant to be used with.

SECTION 5

POLYNOMIAL RNS CONVERSION: THE INNER LEVEL OF PARALLELISM

This section presents an architecture and VLSI design for the conversion into and out of the inner level of parallelism of an algebraic-integer RNS implementation. The conversion, which is based on the polynomial version of the Chinese Remainder Theorem, can be expressed as a matrix multiplication, and the resulting architecture will be based on the LU decomposition of the conversion matrix. This architecture applies to parallel vector-matrix multiplication for any nonsingular matrix, and not just to the special case of the conversion matrix.

The next section describes the inner-level algebraic-integer conversion problem and relates it to LU decompositions. Subsequently, the architecture for the conversion, called the LU architecture, is derived and two different basic cells are treated. An alternative systolic architecture to perform parallel vector-matrix multiplication is considered and compared with the LU architecture. Finally, the VLSI design to perform the LU architecture is described.

5.1 CONVERSION AS MATRIX MULTIPLICATION

5.1.1 Description of the Conversion

In order to make the discussion of the conversion more concrete, it is assumed that the algebraic integers are of the form

$$a_0 + a_1\theta + a_2\theta^2 + \cdots + a_{n-1}\theta^{n-1}, \quad (5.1)$$

where θ is a root of an n th degree polynomial $f(x)$, which is irreducible over the integers and is called the *minimum polynomial* of θ . Other representations for the algebraic integers and the changes that are needed in the conversion are elaborated on at the end of this section.

Let the RNS be composed of the primes p_1, p_2, \dots, p_m . It is possible to extend the results of this section for some cases of arbitrary relatively prime moduli; the restriction of primes is used to ease the exposition. For each prime p , the algebraic integer modulo p must be converted to an n -vector of entries modulo p , where n is the degree of the extension. Processing is then performed on each coordinate independently; this is the inner level of parallelism. Unless otherwise noted, all constants in this section will be residues modulo some fixed prime p .

An algebraic integer of the form of (5.1) modulo a prime p can be considered as an element of the ring $\mathbb{Z}_p[x]/f(x)$, which consists of polynomials with coefficients in \mathbb{Z}_p modulo $f(x)$, i.e., x satisfies the relation $f(x) = 0$. The prime p is chosen so that $f(x)$ has n distinct roots modulo p , say $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$, and so $f(x)$ factors as

$$f(x) = (x - \alpha_0)(x - \alpha_1) \cdots (x - \alpha_{n-1}).$$

From the Chinese Remainder Theorem applied to polynomials, there is the following isomorphism:

$$\begin{aligned} \mathbb{Z}_p[x]/(f(x)) &\cong \mathbb{Z}_p[x]/(x - \alpha_0) \times \cdots \times \mathbb{Z}_p[x]/(x - \alpha_{n-1}) \\ &\cong \mathbb{Z}_p \times \cdots \times \mathbb{Z}_p. \end{aligned}$$

This isomorphism is given by the map

$$\phi: g(x) \mapsto (g(\alpha_0), \dots, g(\alpha_{n-1})). \quad (5.2)$$

Each component of $\phi(g(x))$ represents one of the inner parallel channels; in particular, there is an n -fold increase in parallelism.

If $g(x)$ is represented as in (5.1) (with θ replaced by x), then the mapping in (5.2) is the following modulo p vector-matrix product:

$$(a_0, \dots, a_{n-1}) \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_0 & \alpha_1 & \dots & \alpha_{n-1} \\ \alpha_0^2 & \alpha_1^2 & \dots & \alpha_{n-1}^2 \\ \vdots & \vdots & \dots & \vdots \\ \alpha_0^{n-1} & \alpha_1^{n-1} & \dots & \alpha_{n-1}^{n-1} \end{pmatrix} = (g(\alpha_0), \dots, g(\alpha_{n-1})). \quad (5.3)$$

That is, the conversion to the inner level of parallelism is accomplished by a matrix multiplication. The matrix in (5.3) is an example of a *Vandermonde* matrix. It will be denoted by V . The next theorem proves that the matrix V is invertible (the fact that the roots are distinct is needed here). Conversion out of the inner levels is accomplished by multiplication by V^{-1} .

THEOREM 5.1. *The matrix V is invertible.*

PROOF: For $i = 0, \dots, n-1$, define the following polynomials:

$$v_i(x) = \frac{\prod_{j \neq i} (x - \alpha_j)}{\prod_{j \neq i} (\alpha_i - \alpha_j)}$$

(this is the so-called Lagrange interpolation method). Then $v_i(\alpha_j)$ equals 1 if $i = j$ and 0 otherwise. For each $v_i(x)$, let \mathbf{v}_i be the row vector of coefficients of $v_i(x)$ written by increasing degree. Then $\mathbf{v}_i V = (v_i(\alpha_0), \dots, v_i(\alpha_{n-1}))$ is a vector with a 1 in the i th spot and 0s everywhere else. The matrix whose rows are given by the \mathbf{v}_i is thus the inverse of V . ■

For a general non-prime modulus q , the mapping of (5.2) is an isomorphism exactly when the corresponding Vandermonde matrix is nonsingular. In particular, this is the case if the determinant of the matrix is a unit modulo q . A formula for the determinant of a Vandermonde matrix is given in section 5.1.2.

Note that (5.3) is essentially a transform. Polynomial multiplication or convolution on the left side becomes coordinate-wise multiplication on the right side. Hence, one advantage of the inner level of parallelism is that it greatly simplifies the complicated algebraic-integer multiplication.

5.1.2 LU Decomposition of the Vandermonde Matrix

The architecture to perform the conversion is based on the LU decomposition of the Vandermonde matrix V . A matrix is *LU-decomposable* if it can be factored as the product of a lower-triangular matrix L times an upper-triangular matrix U . Some elementary properties of LU decompositions are given in appendix A.

Suppose first that V factors into two $n \times n$ matrices, say $V = V_1 V_2$. Since V is invertible, V_1 must be invertible. Each row of V_1^{-1} is identified with the vector of coefficients of some polynomial (the coefficients written by increasing degree). With this identification,

$$V_1^{-1} = \begin{pmatrix} - & m_0(x) & - \\ - & m_1(x) & - \\ \vdots & & \\ - & m_{n-1}(x) & - \end{pmatrix}. \quad (5.4)$$

If \mathbf{m}_i is the vector of coefficients for $m_i(x)$, then equations (5.2) and (5.4) give

$$\begin{aligned} (m_i(\alpha_0), \dots, m_i(\alpha_{n-1})) &= \phi(m_i(x)) \\ &= \mathbf{m}_i V \\ &= \mathbf{m}_i V_1 V_2 \\ &= i^{\text{th}} \text{ row of } V_2. \end{aligned}$$

That is,

$$V_2 = \begin{pmatrix} m_0(\alpha_0) & m_0(\alpha_1) & \cdots & m_0(\alpha_{n-1}) \\ m_1(\alpha_0) & m_1(\alpha_1) & \cdots & m_1(\alpha_{n-1}) \\ \vdots & \vdots & & \vdots \\ m_{n-1}(\alpha_0) & m_{n-1}(\alpha_1) & \cdots & m_{n-1}(\alpha_{n-1}) \end{pmatrix}. \quad (5.5)$$

Conversely, if $\{m_0(x), \dots, m_{n-1}(x)\}$ is any basis for $\mathbb{Z}_p[x]$ modulo $f(x)$, then the corresponding matrices V_1 and V_2 from (5.4) and (5.5), respectively, give a factorization of V ; V_1 is a change of basis matrix and V_2 is the matrix for the map in (5.2) in this new basis.

When do V_1 and V_2 give an LU decomposition? If V_1 is lower-triangular, then V_1^{-1} is also lower-triangular, and in particular the degree of $m_i(x)$ is i . If V_2 is upper-triangular, then $m_i(\alpha_j)$ is zero if $j < i$, or equivalently, $(x - \alpha_j)$ divides $m_i(x)$. These observations lead to the following definitions:

$$\begin{aligned} t_0(x) &= 1 \\ t_1(x) &= x - \alpha_0 \\ t_2(x) &= (x - \alpha_0)(x - \alpha_1) \\ &\vdots \\ t_{n-1}(x) &= \prod_{j < n-1} (x - \alpha_j). \end{aligned} \quad (5.6)$$

Define V_1 and V_2 as in (5.4) and (5.5) using these $t_i(x)$ polynomials; this gives the LU decomposition of V . Furthermore, up to a constant factor, they are the only polynomials which do so (Theorem A.3). These matrices are denoted by $L = V_1$ and $U = V_2$. The LU decomposition of a Vandermonde matrix is presented in a different manner in [9] along with additional references.

As an aside, this factorization gives for free the well-known formula for the determinant of a Vandermonde matrix. The determinant of V is the product of the determinants of L and U . The determinant of L is 1, since L^{-1} is lower-triangular with 1s on the diagonal (the coefficient of x^i in $t_i(x)$ is 1). Thus, the determinant of V is the determinant of U , which in turn is the product of its diagonal elements. That is,

$$\det(V) = \prod_{i=0}^{n-1} t_i(\alpha_i) = \prod_{j < i} (\alpha_i - \alpha_j).$$

5.1.3 An Example: $\mathbb{Z}[\sqrt{2+\sqrt{2}}]$, $p = 31$

Let $\theta = \sqrt{2+\sqrt{2}}$. The minimum polynomial for θ is $f(x) = x^4 - 4x^2 + 2$, and a small calculation shows that

$$x^4 - 4x^2 + 2 \equiv (x-5)(x-14)(x-17)(x-26) \pmod{31}.$$

Thus the isomorphism of (5.2) exists. The corresponding Vandermonde matrix is

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 5 & 14 & 17 & 26 \\ 25 & 10 & 10 & 25 \\ 1 & 16 & 15 & 30 \end{pmatrix}.$$

The inverse of V can be found by expanding the following four polynomials (where, for the remainder of this section, all calculations are modulo 31):

$$\begin{aligned} & \frac{1}{(5-14)(5-17)(5-26)} (x-14)(x-17)(x-26) \\ & \frac{1}{(14-5)(14-17)(14-26)} (x-5)(x-17)(x-26) \\ & \frac{1}{(17-5)(17-14)(17-26)} (x-5)(x-14)(x-26) \\ & \frac{1}{(26-5)(26-14)(26-17)} (x-5)(x-14)(x-17). \end{aligned}$$

That is,

$$V^{-1} = \begin{pmatrix} 10 & 2 & 30 & 6 \\ 6 & 27 & 1 & 20 \\ 6 & 4 & 1 & 11 \\ 10 & 29 & 30 & 25 \end{pmatrix}.$$

In order to calculate the LU decomposition of V , define the four polynomials:

$$\begin{aligned} t_0(x) &= 1 &= 1 \\ t_1(x) &= x-5 &= x+26 \\ t_2(x) &= (x-5)(x-14) &= x^2+12x+8 \\ t_3(x) &= (x-5)(x-14)(x-17) &= x^3+26x^2+21x+19. \end{aligned}$$

Then $V = LU$, where

$$L^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 26 & 1 & 0 & 0 \\ 8 & 12 & 1 & 0 \\ 19 & 21 & 26 & 1 \end{pmatrix}$$

(using the $t_i(x)$ in (5.4)) and

$$U = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 9 & 12 & 21 \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

(using the $t_i(x)$ in (5.5)). These in turn imply that

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 25 & 19 & 1 & 0 \\ 1 & 12 & 5 & 1 \end{pmatrix}$$

and

$$U^{-1} = \begin{pmatrix} 1 & 24 & 29 & 6 \\ 0 & 7 & 8 & 20 \\ 0 & 0 & 25 & 11 \\ 0 & 0 & 0 & 25 \end{pmatrix}.$$

The latter two matrices are given for completeness; they will not be needed in the architecture to be described at the end of section 5.2.

This and other examples are given in appendix B.

5.1.4 Change of Basis

An algebraic integer may be naturally represented in several ways. For example, the elements of the ring $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$ of real numbers in the ring of algebraic integers determined by the 16th roots of unity can be represented as

$$(a_0, a_1, a_2, a_3) \longleftrightarrow a_0 + a_1 \sqrt{2 + \sqrt{2}} + a_2 (\sqrt{2 + \sqrt{2}})^2 + a_3 (\sqrt{2 + \sqrt{2}})^3$$

or

$$(b_0, b_1, b_2, b_3) \longleftrightarrow b_0 + b_1 \sqrt{2 + \sqrt{2}} + b_2 \sqrt{2} + b_3 \sqrt{2 - \sqrt{2}}.$$

It is the first representation that coincides with (5.1) and corresponds to the example in section 5.1.3. The two representations are related by a change of basis matrix:

$$(b_0, \dots, b_3) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & -3 & 0 & 1 \end{pmatrix} = (a_0, \dots, a_3).$$

Call this change of basis matrix T . Then the vector (b_0, \dots, b_3) is converted into the inner channels by multiplying by TV , where V is the Vandermonde matrix from (5.3). Since T is lower-triangular and V is LU-decomposable, the matrix TV is LU-decomposable. Therefore, the architecture to be developed in section 5.2 would apply also to the matrix TV and thus to the second representation.

In general, given another representation for the algebraic integers besides the standard representation of (5.1), there is a change of basis matrix T that sends this representation into the standard representation. This matrix is integral with determinant ± 1 , and so in particular is invertible modulo any prime p . The conversion of algebraic integers in the new representation into the inner channels is accomplished by multiplying by TV , where V is the Vandermonde matrix from (5.3). If TV is LU-decomposable, then no additional changes need be made; the resulting matrices L and U can be used in the architecture of section 5.2.

If TV is not LU-decomposable, then there is some permutation matrix P such that PTV is LU-decomposable (see Theorem A.2 in appendix A). Multiplying by the matrix P induces a reordering of the coordinates in the representation, and thus requires no additional overhead. That is, conversion becomes the calculation

$$((a_0, \dots, a_{n-1})P^T) PTV,$$

where multiplying by P^T just permutes the a_i . The same observation applies to reconversion.

5.2 THE LU ARCHITECTURE

In this section, an architecture is developed that performs the conversion and reconversion of a vector via a matrix multiplication. The architecture applies whenever the matrix is LU-decomposable and is called the *LU architecture*. The principle used in the computation is a combination of forward elimination and back substitution (see [9]). The architecture is also motivated by the desire to have parallel inputs and outputs.

The LU architecture assumes knowledge of the LU decomposition; as seen in the last section, it is possible to precompute the LU decomposition for the the case of

inner-level conversion and reconversion. In other applications, determining the LU decomposition may be more the central issue, and the present architecture should not be confused with methods that accomplish this. This section develops the LU architecture in a general manner, illustrating it using the specific task of inner-level conversion and reconversion.

5.2.1 Data Flow

Suppose that the vector $\mathbf{x} = (x_0, \dots, x_{n-1})$ is converted to the vector $\mathbf{z} = (z_0, \dots, z_{n-1})$ using an LU-decomposable matrix. Then for the appropriate lower-triangular matrix L and upper-triangular matrix U ,

$$\mathbf{x}L\mathbf{U} = \mathbf{z}. \quad (5.7)$$

Reconversion from \mathbf{z} to \mathbf{x} is the matrix product

$$\mathbf{z}U^{-1}L^{-1} = \mathbf{x}. \quad (5.8)$$

Define $\mathbf{y} = (y_0, \dots, y_{n-1})$ as $\mathbf{x}L$ ($= \mathbf{z}U^{-1}$). This \mathbf{y} is in some sense a halfway point in the conversion from \mathbf{x} to \mathbf{z} (or the reconversion the other way).

Although the results in this section will be discussed for general LU decompositions, the specific example of the Vandermonde matrix of the last section should be kept in mind. In that case, the vector \mathbf{x} represents an algebraic integer in polynomial notation (5.1). The vector \mathbf{z} represents residues (from the polynomial version of the Chinese Remainder Theorem (5.2)). As for \mathbf{y} , it is the vector of coefficients of the $t_i(x)$ from (5.6). The $t_i(x)$ are products of an increasing number of irreducible polynomials, i.e., the $(x - \alpha_i)$, and so are analogous to the successive products of primes in mixed-radix conversion in ordinary-integer RNS; the vector \mathbf{y} then corresponds to mixed-radix coefficients. (This analogy can be made rigorous; it is based on the similar algebraic properties of \mathbb{Z} and $\mathbb{Z}_p[x]$.)

The overall structure of the LU architecture is a grid of basic cells that uses the fact that since L is lower-triangular, y_i depends only on x_i, \dots, x_{n-1} , or equivalently, on $x_i, y_{i+1}, \dots, y_{n-1}$. As each y_i is calculated by forward elimination, it is then fed into the calculation of y_0, \dots, y_{i-1} and z_i, \dots, z_{n-1} . The latter are computed by back substitution. The flow of variables is shown in figure 16. Values pass from top to bottom and calculations take place at the arrowheads. In particular, the grid in figure 16 is 5×4 . This process will be made more apparent in the next subsection.

The reconversion is based on similar observations between the y_i and the z_i . Since U^{-1} is upper-triangular, y_i depends on z_0, \dots, z_i , i.e., on z_i, y_0, \dots, y_{i-1} . Again after being calculated by back substitution, each y_i is then used to calculate y_{i+1}, \dots, y_{n-1} .

and x_0, \dots, x_i . This flow of variables is shown in figure 17. Observe that though the underlying grid is identical to that in figure 16, the vectors are written in reverse order.

The LU architecture can be applied to matrix-matrix multiplication by pipelining the input vectors. The only requirement is that one of the matrices be nonsingular.

As already stated, in order to complete the architectures in figures 16 and 17, some type of computation is performed at the arrowheads. These locations are called *cells*. The function of these cells is the choice of the designer and could depend on implementation, etc. The next two subsections each deal with one type of cell.

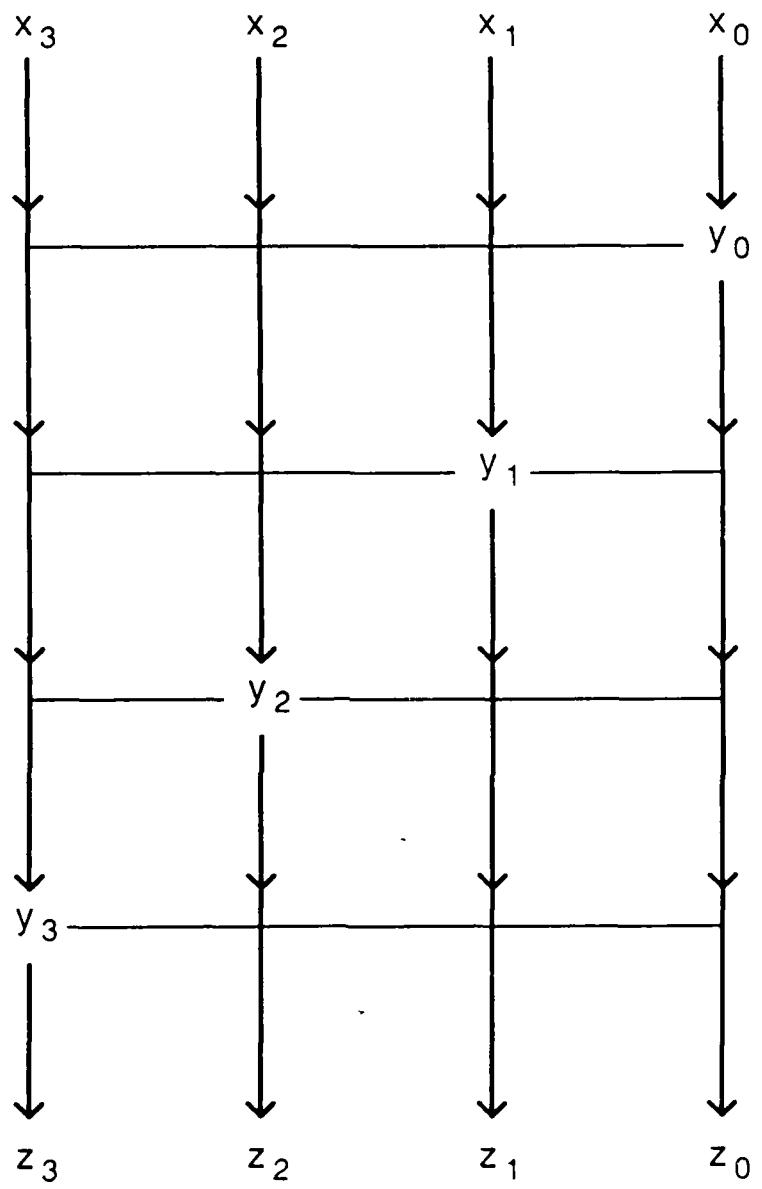


Figure 16. Flow of Variables in LU Grid ($n = 4$)

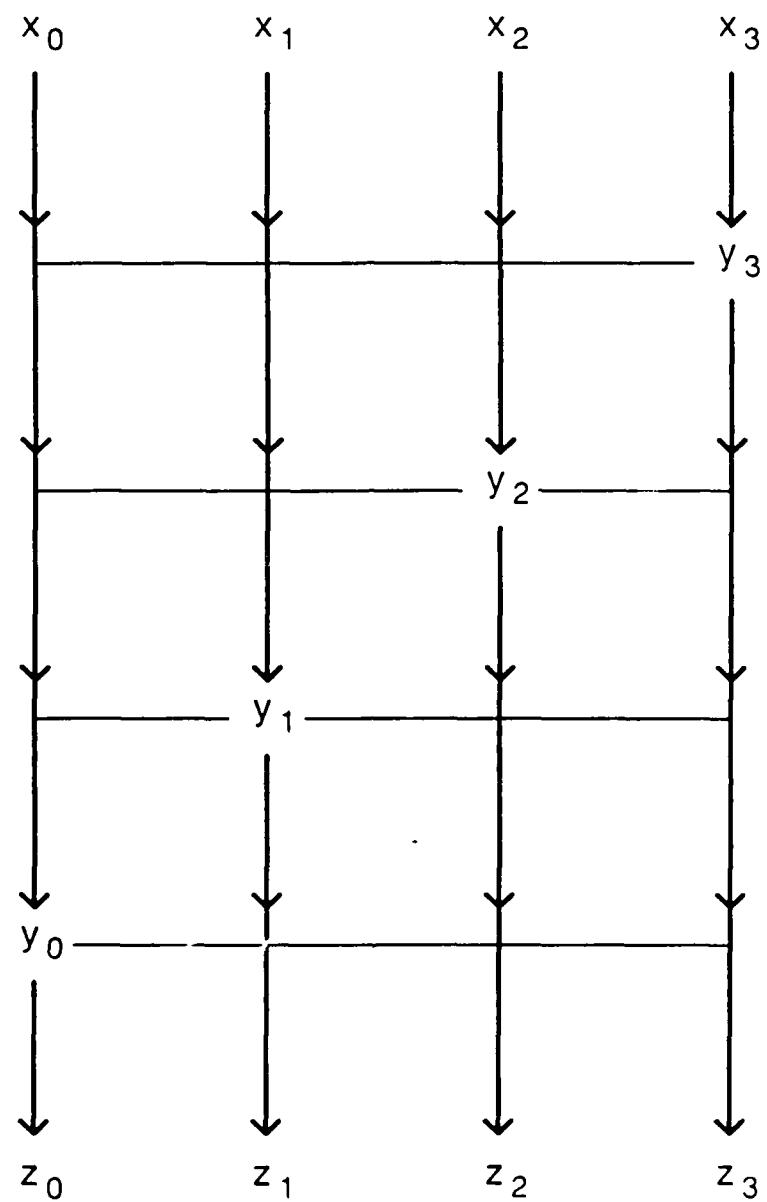


Figure 17. Flow of Variables in UL Grid ($n = 4$)

5.2.2 The $A + BC$ Cell

The first cell is shown in figure 18. The operation of the cell is a multiply and accumulate. There are two inputs, A and B , which are used to produce the result $A + BC$, where C is a constant assumed fixed for the cell, i.e., stored in the cell. In the algebraic-integer conversion, the calculation is carried out modulo some fixed prime p .

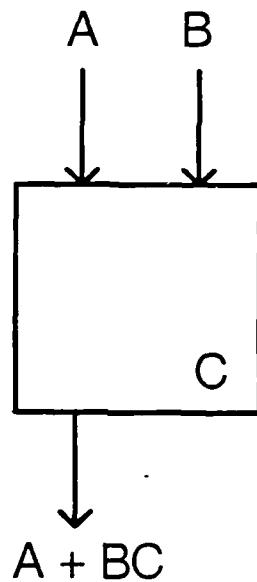


Figure 18. The $A + BC$ Cell

The following observation from appendix A is needed: if a matrix has LU decomposition LU , then for any nonsingular diagonal matrix D , $(LD^{-1})(DU)$ is also an LU decomposition. In this way, the diagonal of L or U can be prescribed as required.

5.2.2.1 Architecture for LU

The matrices of the LU decomposition are written so that the diagonal of L is all 1s (and so the diagonal of L^{-1} is also all 1s). To ease the exposition, let $n = 4$ and let

$$L^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \ell_{10} & 1 & 0 & 0 \\ \ell_{20} & \ell_{21} & 1 & 0 \\ \ell_{30} & \ell_{31} & \ell_{32} & 1 \end{pmatrix}$$

and

$$U = \begin{pmatrix} u_{00} & u_{01} & u_{02} & u_{03} \\ 0 & u_{11} & u_{12} & u_{13} \\ 0 & 0 & u_{22} & u_{23} \\ 0 & 0 & 0 & u_{33} \end{pmatrix}.$$

Since $yL^{-1} = x$,

$$\begin{aligned} y_3 &= x_3 \\ y_2 &= x_2 - \ell_{32}y_3 \\ y_1 &= x_1 - \ell_{31}y_3 - \ell_{21}y_2 \\ y_0 &= x_0 - \ell_{30}y_3 - \ell_{20}y_2 - \ell_{10}y_1. \end{aligned} \tag{5.9}$$

These equations determine the constants for the triangle of cells above the off diagonal (see figure 19). The remaining cells are filled with the appropriate u_{ij} ; this follows from the equation $z = yU$. By choosing the diagonal of L to be all 1s, the coefficient of x_i is also 1, and so the first row of figure 16 can be eliminated.

In the example of section 5.1.3, the matrices L^{-1} and U are already in the required form. The resulting grid with constants is shown in figure 20.

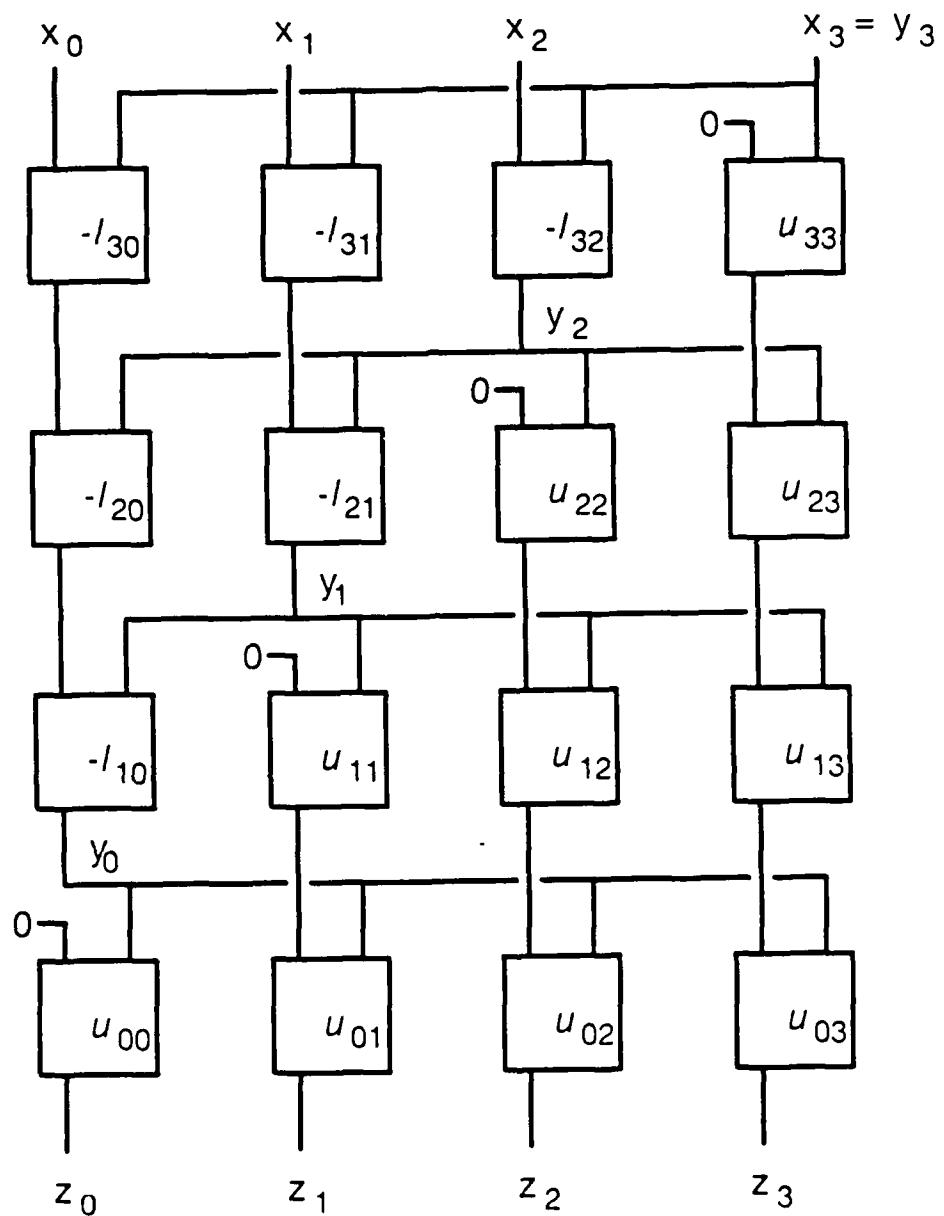


Figure 19. Grid for LU Computation with $A + BC$ Cell

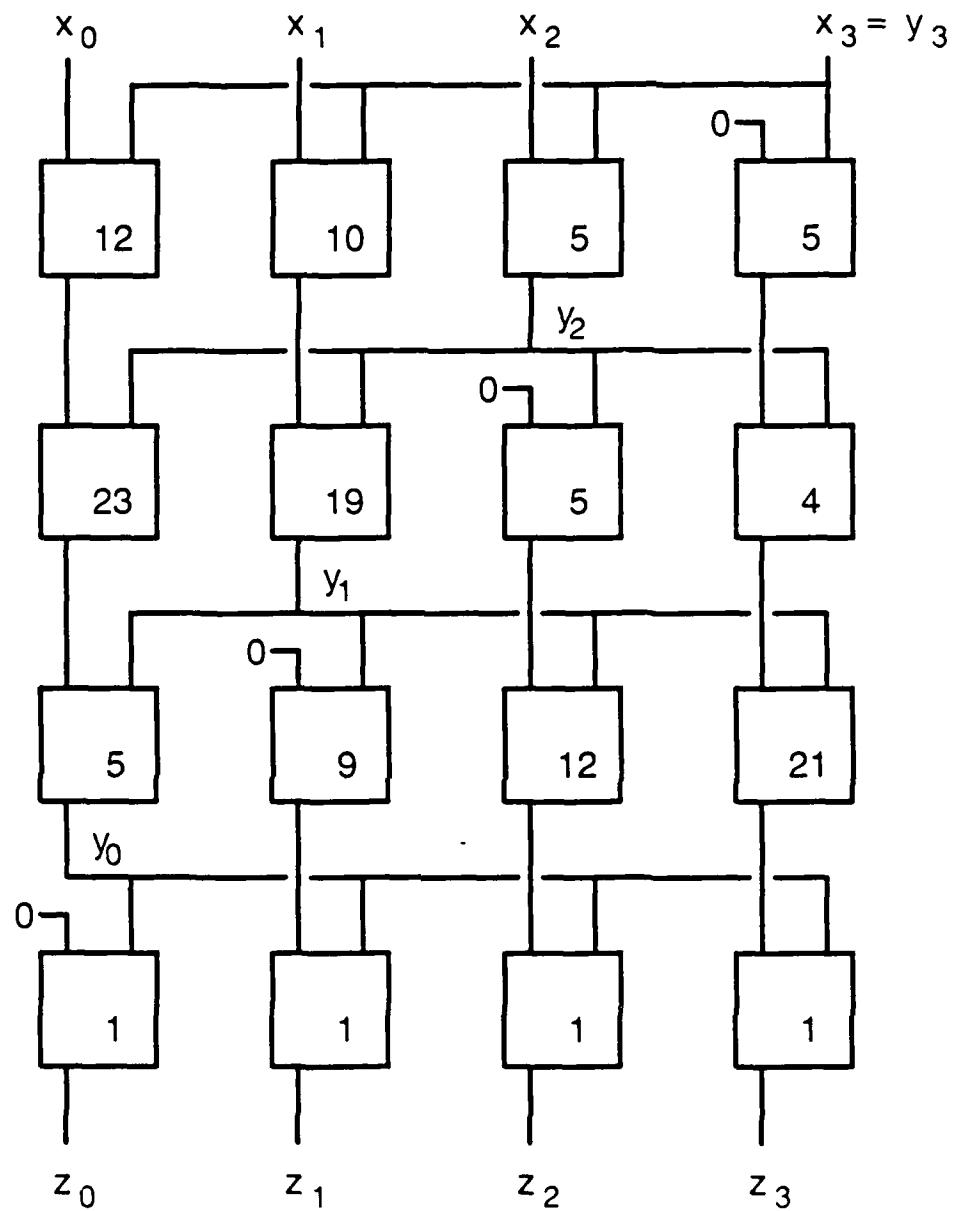


Figure 20. A Specific Example of LU Computation

5.2.2.2 Architecture for UL

The matrices of the LU decomposition are written so that the diagonal of U is all 1s. Continuing with $n = 4$, let

$$L^{-1} = \begin{pmatrix} \ell_{00} & 0 & 0 & 0 \\ \ell_{10} & \ell_{11} & 0 & 0 \\ \ell_{20} & \ell_{21} & \ell_{22} & 0 \\ \ell_{30} & \ell_{31} & \ell_{32} & \ell_{33} \end{pmatrix}$$

and

$$U = \begin{pmatrix} 1 & u_{01} & u_{02} & u_{03} \\ 0 & 1 & u_{12} & u_{13} \\ 0 & 0 & 1 & u_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

(Of course, the l_{ij} and u_{ij} may be different from those in the previous section.)

Since $\mathbf{y}U = \mathbf{z}$,

$$\begin{aligned} y_0 &= z_0 \\ y_1 &= z_1 - u_{01}y_0 \\ y_2 &= z_2 - u_{02}y_0 - u_{12}y_1 \\ y_3 &= z_3 - u_{03}y_0 - u_{13}y_1 - u_{23}y_2. \end{aligned} \tag{5.10}$$

These equations determine the constants for the triangle of cells above the off diagonal (see figure 21). Notice that the vectors are reversed from figure 19. The remaining cells are filled with the appropriate ℓ_{ij} ; this follows from the equation $\mathbf{x} = \mathbf{y}L^{-1}$. Again, the first row of cells in figure 17 was eliminated by the choice of diagonal for U .

In the example of section 5.1.3, the matrices L^{-1} and U must be changed so that the diagonal of U is all 1s. The new matrices are

$$L^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 27 & 7 & 0 & 0 \\ 14 & 21 & 25 & 0 \\ 10 & 29 & 30 & 25 \end{pmatrix}$$

and

$$U = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 22 & 23 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The resulting grid is shown in figure 22.

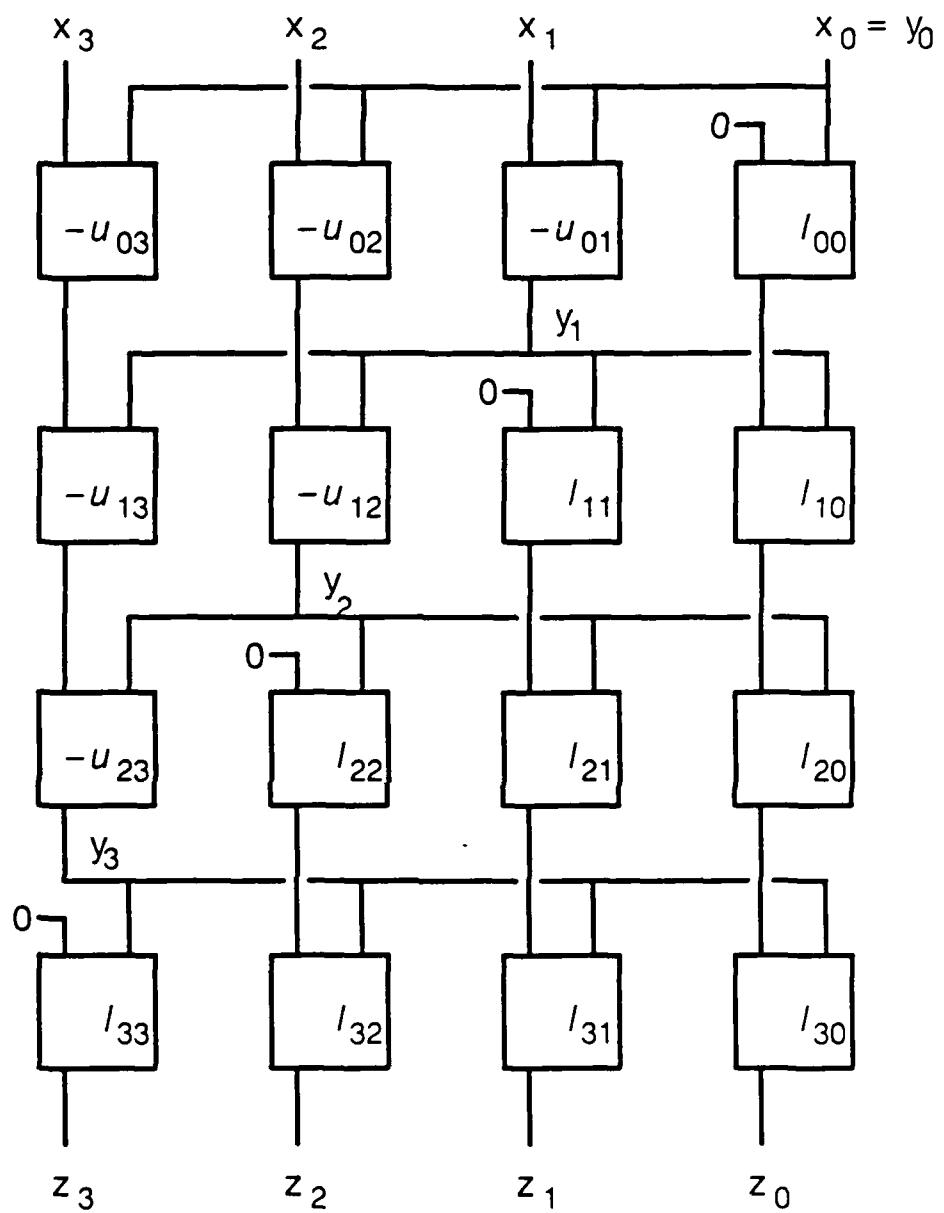


Figure 21. Grid for UL Computation with $A + BC$ Cell

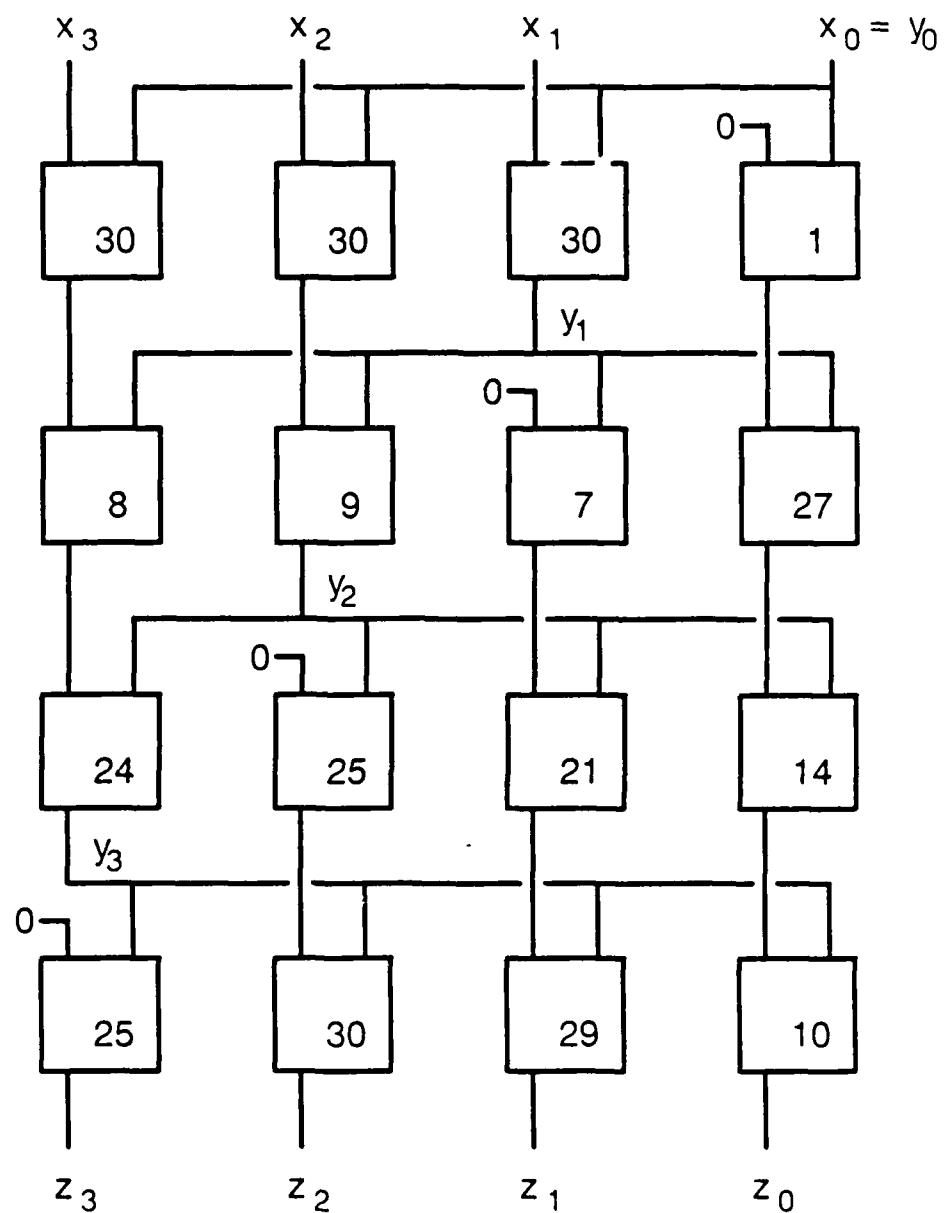


Figure 22. A Specific Example of UL Computation

5.2.3 The $C(A - B)$ Cell

The $A + BC$ cell from above gives a square grid for any LU decomposition. Because of the analogy with ordinary-integer mixed-radix conversion, another cell seems natural, namely one which does a subtraction and then a multiplication. This is shown in figure 23. The inputs A and B are first subtracted and the result is multiplied by a (stored) constant C .

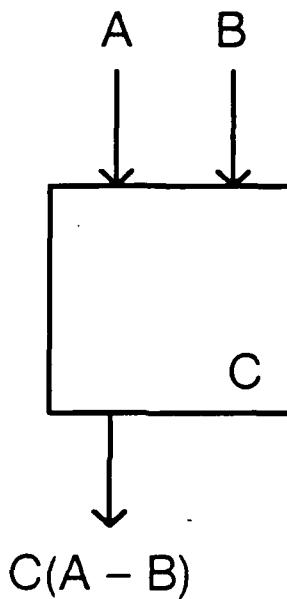


Figure 23. The $C(A - B)$ Cell

Unfortunately, this cell does not lead to a well-defined architecture for general LU decompositions. Consider the equations in (5.9), specifically

$$y_1 = x_1 - \ell_{31}y_3 - \ell_{21}y_2.$$

In order to put this in the grid, there would need to be constants β_j such that the following equation holds

$$y_1 = \beta_3(\beta_2(\beta_1(x_1 - 0) - y_3) - y_2). \quad (5.11)$$

Equating coefficients yields the equations

$$\begin{aligned}\beta_3\beta_2\beta_1 &= 1 \\ \beta_3\beta_2 &= \ell_{31} \\ \beta_3 &= \ell_{21}.\end{aligned}$$

In particular, $\beta_2 = \ell_{31}/\ell_{21}$. But there is no guarantee that ℓ_{21} is nonzero. The same problem occurs with the equations for reconversion (5.2). One can attempt to change the order of operations in (5.3) or to change the diagonal of the respective matrix, but the same criticism will still hold. In this way, there is no well-defined method to assign constants to the cells for an arbitrary LU decomposition (for either conversion or reconversion).

In the algebraic-integer conversion, there are such constants β_j for the U matrix because the i, j entry

$$\prod_{k < i} (\alpha_j - \alpha_k)$$

factors in a natural way. This also can be seen from the analogy with ordinary-integer mixed-radix conversion. The lower-triangular matrix though does not work in general. It is straightforward, however, to combine the $C(A - B)$ cells for U with $A + BC$ cells for L . The resulting grid for the UL case is analogous to ordinary-integer mixed-radix conversion. These hybrid grids though are undesirable because of the lack in regularity; this is especially so since all additions and multiplications are modulo the same prime. Therefore, the $A + BC$ cell is the better choice.

The grids using the $C(A - B)$ cell are not shown; if they exist they would be similar to figures 19 and 21.

5.3 SYSTOLIC CONVERSION

The LU architecture developed in section 5.2 computes in parallel a modulo p vector-matrix product, but only if the matrix is LU-decomposable (equations (5.1) and (5.2)). By using Theorem A.2 in appendix A, this architecture applies to an arbitrary nonsingular matrix after an appropriate permutation of the coordinates of the incoming vector. It is natural though to examine other methods to compute vector-matrix products, and one common approach is to perform the computation via a systolic architecture. A complete treatment of systolic architectures is beyond the scope of this paper. Rather the LU architecture is modified into a systolic architecture. The architectures are linked by the common use of the $A + BC$ cells.

5.3.1 The Systolic Architecture

A *systolic* architecture is characterized by computation taking place in a set of locally interconnected simple cells. It is desirable for the computation to be pipelined so that the cells are in constant use. The LU architecture is *semi-systolic*; it is fully pipelineable and the operations are performed in simple cells. Information, however, must be made available globally via long buses. The goal then is to eliminate these long buses. For a general introduction to systolic architectures, see [11].

It should be stressed that for certain applications there are more suitable architectures than those discussed here. For example, if the n inner channels were implemented with a single channel running n times as fast, the n coordinates would be available sequentially. In this case, a linear systolic array, requiring only n as opposed to n^2 cells, may be more appropriate.

If the inputs and outputs are to be processed in parallel, then systolic arrays to perform matrix-matrix multiplication are required. There are a variety of systolic architectures that perform matrix-matrix multiplication involving a host of basic cells and data-flow strategies ([17]). When the restriction is made to use the $A + BC$ cell described in section 5.2 (figure 18) and a rectangular data flow, then the systolic architecture of this section is suggested.

The systolic architecture is described for $n = 4$, and is easily applied for general n . Suppose that the goal is to compute the vector-matrix product $\mathbf{x}V$, i.e.,

$$(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \begin{pmatrix} v_{00} & v_{01} & v_{02} & v_{03} \\ v_{10} & v_{11} & v_{12} & v_{13} \\ v_{20} & v_{21} & v_{22} & v_{23} \\ v_{30} & v_{31} & v_{32} & v_{33} \end{pmatrix}.$$

The flow of data is shown in figure 24. Each matrix element resides in a basic cell, so there are n^2 cells. The inputs enter from the left and are staggered. They are passed unchanged along the horizontal arrows. Each input is multiplied by the matrix element and the result accumulated and passed down to the next cell along the vertical arrows. For example, at step 1, $\mathbf{x}_0 v_{00}$ is computed. At step 2, $\mathbf{x}_0 v_{01}$ and $\mathbf{x}_1 v_{10}$ are computed, with the latter then added to the previously computed $\mathbf{x}_0 v_{00}$. After step 3, the partial answers are $\mathbf{x}_0 v_{02}$, $\mathbf{x}_0 v_{01} + \mathbf{x}_1 v_{11}$, and $\mathbf{x}_0 v_{00} + \mathbf{x}_1 v_{10} + \mathbf{x}_2 v_{20}$. The outputs thus come out staggered at the bottom at steps 4, 5, 6, and 7.

This process is further illustrated in figure 25. Each cell is an $A + BC$ cell of figure 19. The result is fully pipelineable, as is illustrated by the vector \mathbf{y} following the vector \mathbf{x} . Once the pipe is filled, a set of four coordinates, corresponding to four different output vectors, is available at each step.

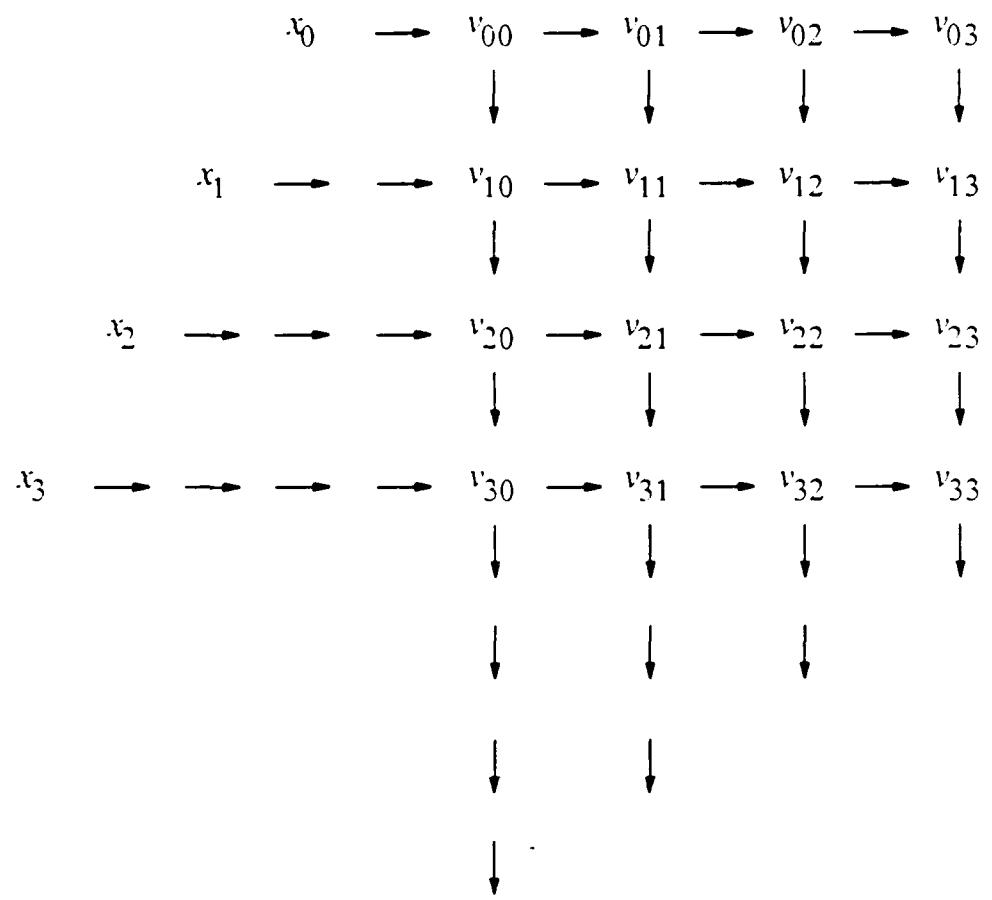


Figure 24. Data Flow in the Systolic Architecture

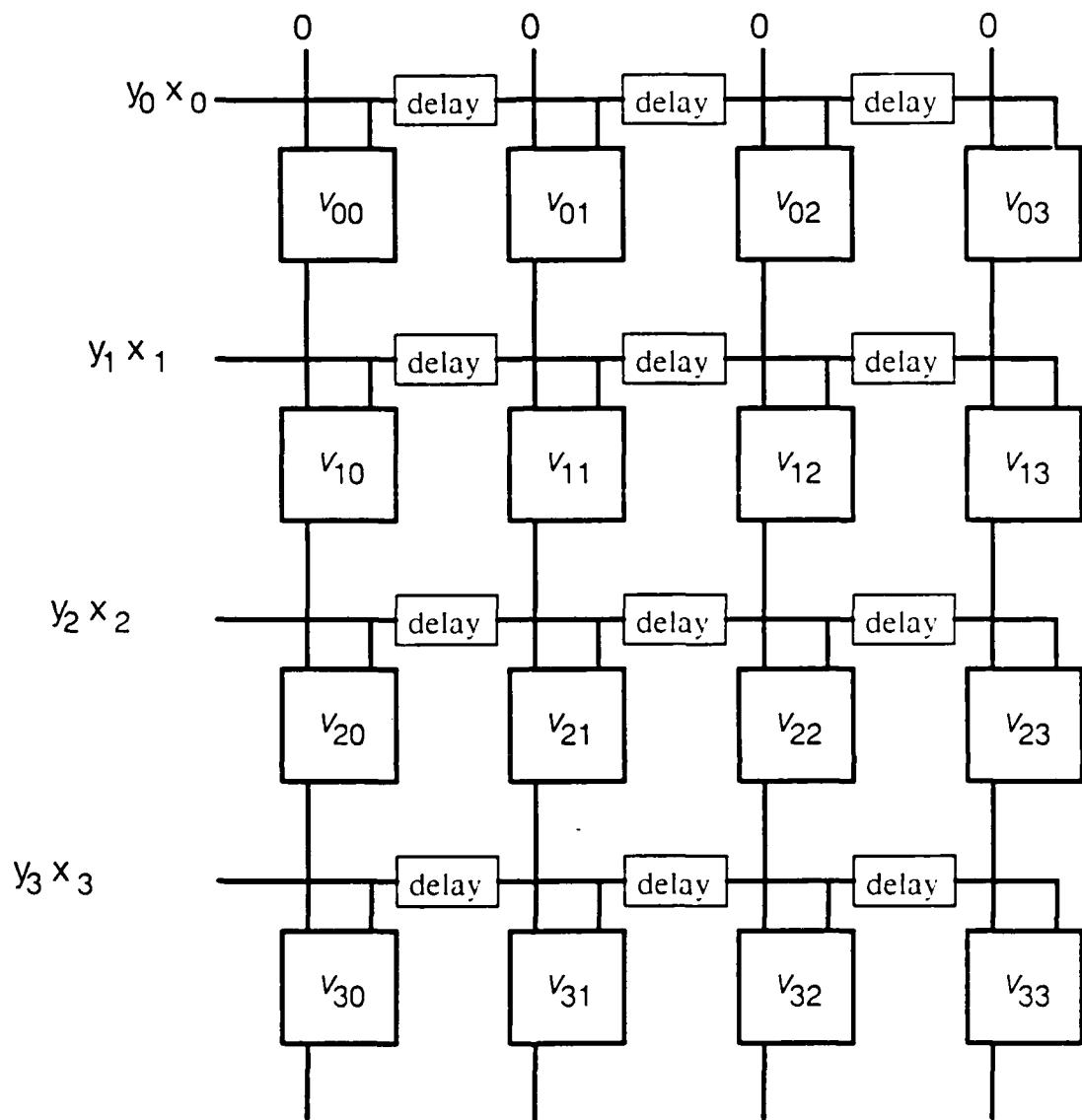


Figure 25. The Systolic Architecture with $A + BC$ Cells

5.3.2 Comparison with LU Architecture

Both the LU architecture and the systolic architecture are fully pipelineable. Both can be used to compute in parallel a modulo p vector-matrix multiplication where the matrix is arbitrary (for the systolic architecture), or arbitrary nonsingular (for the LU architecture, after possibly rearranging inputs). Both architectures are easily partitioned into smaller blocks, say by rows or 2×2 subgrids. For the systolic architecture, these blocks are identical; for the LU architecture, a slight modification must be made depending on whether certain lines are input or output lines.

The major difference between the two architectures is the timing of the inputs and outputs. The systolic architecture requires the inputs and outputs to be staggered, and thus after the pipe is full the n coordinates entering and exiting the converter at each time step correspond to n consecutive algebraic-integer results. Therefore, the systolic architecture requires more complicated synchronization before and after the converter. On the other hand, the n coordinates entering and leaving the converter for the LU architecture correspond to a single algebraic-integer result. However, long buses are needed at each time step. This may be a concern for larger matrices.

5.4 VLSI IMPLEMENTATION

Both the LU architecture of section 5.2 and the systolic architecture of section 5.3 are highly regular, and thus well suited to VLSI implementation. In such an implementation, the architecture would be divided up evenly into chips, with as much of the grid of the architecture put onto a single chip as feasible.

Although a single chip implementing both architectures is possible, since both use the same basic $A + BC$ cell, the chip to be fabricated will implement the LU architecture. This architecture was chosen over the systolic architecture because of I/O considerations, namely the fact that the algebraic-integer vectors do not need to be staggered. Furthermore, the long buses required are not a problem for the degrees being considered (up to degree 8). The systolic architecture is not implemented along with the LU architecture because of the added complexity in the design due to the extra delays needed. The I/O assignments on the chip are not ideally suited to both architectures. It should not be difficult, however, in adapting the actual VLSI layout to a systolic approach if such is desired in the future.

Recall that the grid for the LU architecture consists of an $n \times n$ square of $A + BC$ cells. Furthermore, for the applications of algebraic integers usually considered, n is even, and this will be the assumption here.

5.4.1 Chip Layout Limitation

The chip to be fabricated uses an 84-pin package and a 7.9-mm \times 9.2-mm die size. These parameters were chosen based on two main limitations derived from the architecture.

The first major limitation to the VLSI implementation of the LU architecture is modulus programmability. Specifying a prime or family of primes beforehand limits the possible sets of algebraic integers. Instead, the prime needed in the conversion should be an input (though fixed during a given application). This ability to program the prime requires a larger area to perform the modulo arithmetic. This in turn places limits on the VLSI chips, since a chip consists mainly of $A + BC$ cells, each of which consists of a modulo adder and multiplier.

The other major limitation to the VLSI implementation is the number of input and output lines, under the assumption that the values in the architectures are to be bused in parallel. Area estimates for the $A + BC$ cells suggested that four of these cells would fit on a chip. There are three natural ways to place four cells in an $n \times n$ grid: in a row, in a column, and as a 2×2 sub-grid (recall figure 19). The I/O requirements for just the cells, i.e., ignoring for the moment the prime, the cell constants, etc., are 9 lines for the row, 7 lines for the column, and 6 lines for the 2×2 sub-grid. Here the lines are b -bits wide, with b to be determined. Notice further that the first two placements require that n be divisible by 4. Also, the 2×2 placement corresponds itself to the grid for a degree 2 extension (such as QRNS). Hence, placement in a 2×2 sub-grid is suggested.

The prime and constants could be read in by a single 1-bit line. However, allowing wider (b -bit) lines would increase the generality of the chip, letting, for example, the matrix be changed during processing (though this may reduce the speed of the chip). Allowing one b -bit line for the prime and two other b -bit lines for the cell constants would bring the total to 9 lines for the 2×2 sub-grid. For an 84-pin package, the implied upper limit on b is 8. Setting $b = 8$, there would then be 12 remaining pins to allow for the control lines, the clocks, and so forth.

5.4.2 Algorithms for the $A + BC$ Cell

The computation in an $A + BC$ cell consists of a multiplication and an accumulation modulo an 8-bit modulus. The accumulation can be absorbed in the multiplication, so that the two major operations performed in the cell are the computation of $A + BC$ and the reduction of the result modulo the modulus. (It should be noted that nothing inherent in the design requires that the modulus be a prime.)

The multiplication uses a standard array multiplier, see, for example, [8]. Such a multiplier consists of an 8×8 square grid of small cells, each of which is made up of a full adder and an *and* gate. By setting the initial sums to the bits of A , rather than zero, the multiplier computes $A + BC$, rather than just BC . This result is 16 bits long, assuming that all of the input values are 8 bits.

The multiplication/accumulation is followed by a modulo reduction, which is essentially a division. The 16-bit number N is divided by the 8-bit modulus M to produce a quotient Q , which could be 9 bits, and an 8-bit remainder R . At each successive stage, N is updated based on a comparison with a shift of M . If, for the current value of N , $N \geq 2^i M$, then the i th bit of Q is 1 and N is decremented by $2^i M$; otherwise N is unchanged and the i th bit of Q is 0. Here i runs from 8 down to 0. After the last loop, the value of N is precisely R . In VLSI, the inequality $N \geq 2^i M$ is checked by adding $2^{16} - 2^i M$ to N and checking for a carry. It is then straightforward to keep the old value of N or the new sum, which is N decremented by $2^i M$, based on the value of the carry. The carry is, in fact, the i th bit of Q . This algorithm is demonstrated in figure 26.

In practice, only 8-bit adders, not 16-bit adders, are needed to perform this addition. This is because $2^{16} - 2^i M = 2^i(2^{16-i} - M)$ and $2^{16-i} - M$ is just $2^8 - M$ with $8 - i$ leading ones. Hence, the 8-bit number $2^8 - M$ is added to the appropriate 8 bits of N . The carry of the 16-bit addition is just the next higher bit of N *or-ed* with the carry of this 8-bit addition. All of the more significant bits of N are necessarily 0, and the bits of N below the 8 bits used in the addition are not needed (and are just passed down). The example of figure 26 is shown in figure 27 using these observations. Here the 8-bit addition is shown in the box. The next higher bit of N is shown to the left, and the unused lower bits of N to the right.

$$N = 35,161 \quad M = 199$$

Step 1	Current N $2^{16} - 2^8 M$	<u>1000 1001 0101 1001 0011 1001 0000 0000</u>	Carry = 0
Step 2	Current N $2^{16} - 2^7 M$	<u>1000 1001 0101 1001 1001 1100 1000 0000</u>	Carry = 1
Step 3	Current N $2^{16} - 2^6 M$	<u>0010 0101 1101 1001 1100 1110 0100 0000</u>	Carry = 0
Step 4	Current N $2^{16} - 2^5 M$	<u>0010 0101 1101 1001 1110 0111 0010 0000</u>	Carry = 1
Step 5	Current N $2^{16} - 2^4 M$	<u>0000 1100 1111 1001 1111 0011 1001 0000</u>	Carry = 1
Step 6	Current N $2^{16} - 2^3 M$	<u>0000 0000 1000 1001 1111 1001 1100 1000</u>	Carry = 0
Step 7	Current N $2^{16} - 2^2 M$	<u>0000 0000 1000 1001 1111 1100 1110 0100</u>	Carry = 0
Step 8	Current N $2^{16} - 2^1 M$	<u>0000 0000 1000 1001 1111 1110 0111 0010</u>	Carry = 0
Step 9	Current N $2^{16} - 2^0 M$	<u>0000 0000 1000 1001 1111 1111 0011 1001</u>	Carry = 0

$$R = 0000 0000 1000 1001 = 137$$

$$Q = 0 1011 0000 = 176$$

$$35,161 = 176 \times 199 + 137$$

Figure 26. A Modulo Reduction Example (16-bit Adders)

$$N = 35,161 \quad M = 199$$

Step 1	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table>	1	0	0	0	1	0	0	1	0	0	1	1	1	0	0	1	0 1 0 1 1 0 0 1	Result of OR = 0
1	0	0	0	1	0	0	1												
0	0	1	1	1	0	0	1												
Step 2	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table>	1	0	0	0	1	0	0	1	0	0	1	1	1	0	0	1	1 0 1 1 0 0 1	Result of OR = 1
1	0	0	0	1	0	0	1												
0	0	1	1	1	0	0	1												
Step 3	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table>	0	1	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0 1 1 0 0 1	Result of OR = 0
0	1	0	0	1	0	1	1												
0	0	1	1	1	0	0	1												
Step 4	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	1	0	0	1	0	1	1	0	0	0	1	1	0	0	1	1	1 1 0 0 1	Result of OR = 1
1	0	0	1	0	1	1	0												
0	0	1	1	0	0	1	1												
Step 5	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	1	1	0	0	1	1	1	0	0	1	1	0	0	1	1	1 0 0 1	Result of OR = 1
0	1	1	0	0	1	1	1												
0	0	1	1	0	0	1	1												
Step 6	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	0	1	0	0	1	0	0	1	1	0	0	1	1	0 0 1	Result of OR = 0
0	0	0	0	1	0	0	1												
0	0	1	1	0	0	1	1												
Step 7	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	1	0	0	0	1	0	0	1	1	0	0	1	1	0 1	Result of OR = 0
0	0	0	1	0	0	0	1												
0	0	1	1	0	0	1	1												
Step 8	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	0	0	0	1	0	0	0	1	1	0	0	1	1	1	Result of OR = 0
0	0	1	0	0	0	1	0												
0	0	1	1	0	0	1	1												
Step 9	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	1	0	0	0	1	0	1	0	0	1	1	0	0	1	1		Result of OR = 0
0	1	0	0	0	1	0	1												
0	0	1	1	0	0	1	1												

$$R = 1000\ 1001 = 137$$

$$Q = 0\ 1011\ 0000 = 176$$

$$35,161 = 176 \times 199 + 137$$

Figure 27. A Modulo Reduction Example (8-bit Adders)

5.4.3 Layout and Testing

This section contains some pictures of the chip. The chip was designed using scalable CMOS technology. Figure 28 shows the multiplier/accumulator. The 8 bits of C come from the right; the 8 bits of A and B come in from the bottom. The 16 bits of $A + BC$ come out from the left and top. Figure 29 shows the modulo reduction. The 16 bits of the input come in from the top and right; the remainder exits from the bottom. There are 9 rows, each an 8-bit adder. After every three rows, there is a set of registers for pipelining (the last set is not shown, since it is outside this cell). The registers allow for pipelining; each section isolated by registers takes roughly the same time as the multiplier. Figure 30 shows the complete $A + BC$ cell, with the modulo reduction on the left and the multiplier on the right. There is a set of registers between the two. Finally, figure 31 shows a picture of the whole chip with four cells forming a 2×2 grid.

Currently, the chip is undergoing testing before fabrication. SPICE simulations for the basic full adder used in both the multiplier and the modulo reducer indicate a speed of roughly 3 MHz. The chip will be fabricated using 3 μ m technology.

5.5 CONCLUSION

The conversion into and out of the inner level of parallelism of an algebraic-integer implementation is a modulo p vector-matrix multiplication. When the algebraic integers are in a natural representation (eq. 5.1), the conversion matrix is a Vandermonde matrix. Some facts were derived about the Vandermonde matrix, including its LU factorization. This latter result was used in the illustration of an architecture to perform the conversion.

This architecture, called the LU architecture, uses the LU factorization of a matrix to perform vector-matrix multiplication. With some slight modification, it can be used with any nonsingular matrix. The LU architecture is semisystolic, in that computations are performed in a grid of small basic cells. Unfortunately, long buses are needed to distribute information globally. The LU architecture was compared to a systolic architecture using the same basic cells. Although with the systolic architecture all information is just distributed locally, the input and output vectors need to be staggered. This is not the case with the LU architecture, which requires these vectors to be available all at once. Because of this feature, the LU architecture was chosen for implementation in VLSI.

A degree n conversion using the LU architecture requires an $n \times n$ grid of basic cells. For the VLSI implementation, this grid is blocked off into $n/2 \times n/2$ grid of chips, each chip containing a 2×2 grid of basic cells (n is assumed even). The chip is limited to an 8-bit modulus, so that all values are also limited to 8 bits. The matrix can either

be fixed or changed during computation, although the latter may reduce speed. The chip has been designed using scalable CMOS technology. SPICE simulations indicate a speed of 3 MHz. The chip is currently undergoing testing before fabrication using 3 μ m technology.

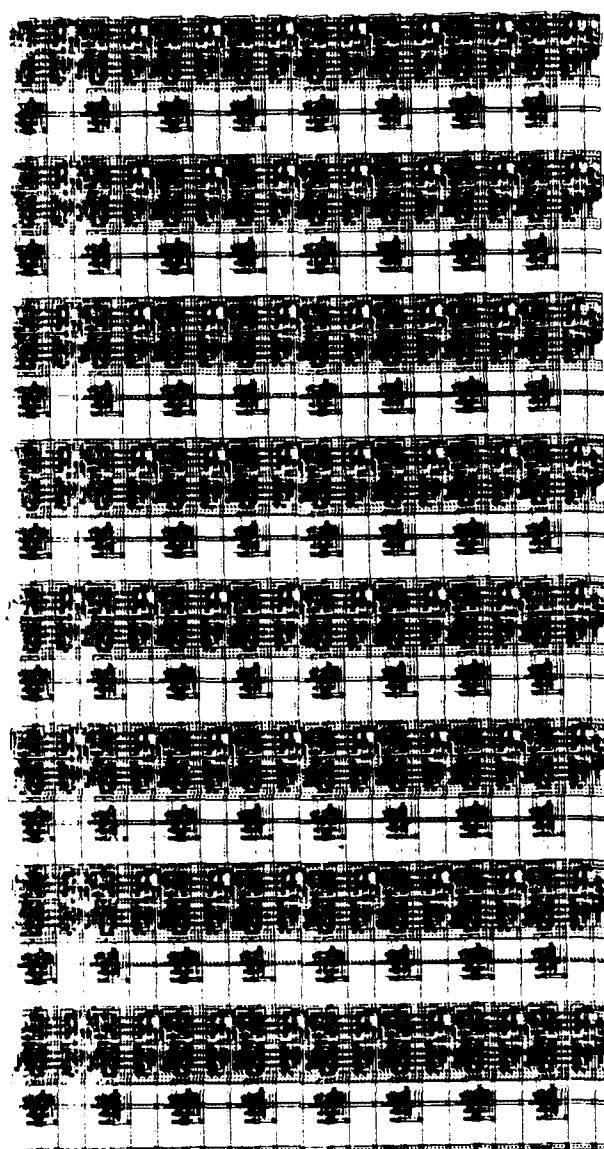


Figure 28. The Multiplier/Accumulator

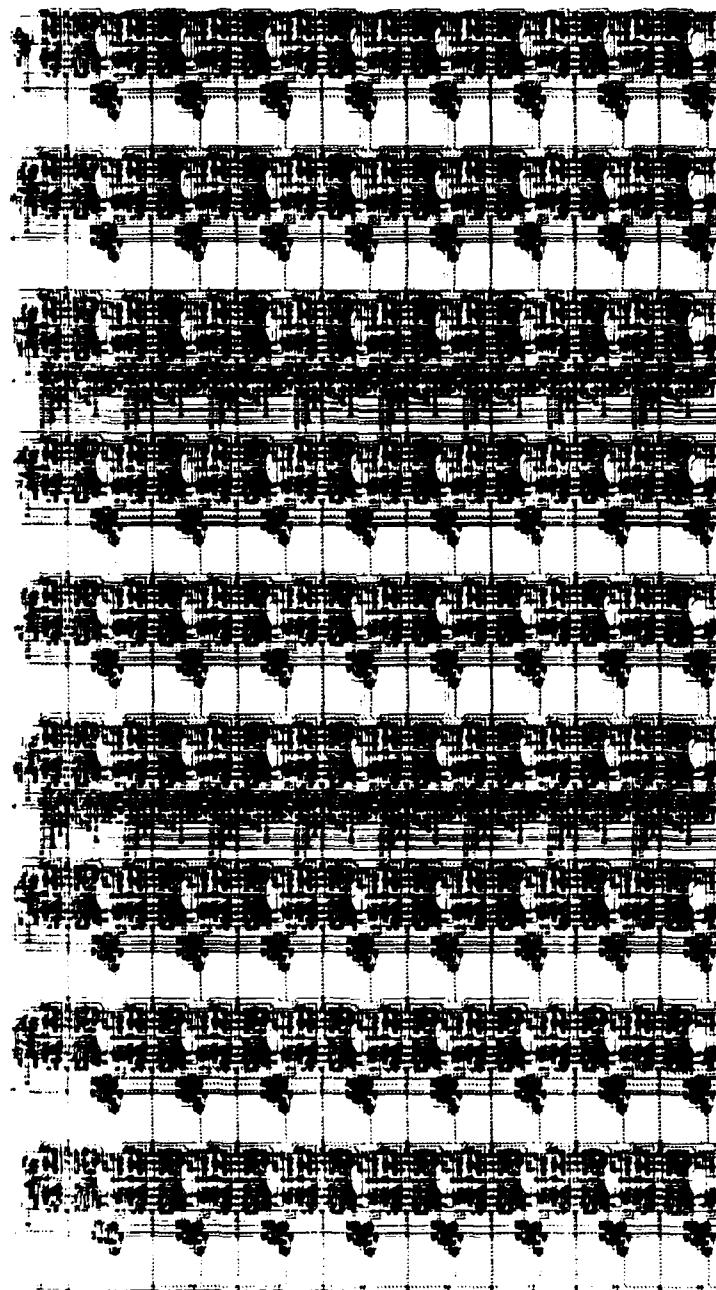


Figure 29. The Modulo Reduction

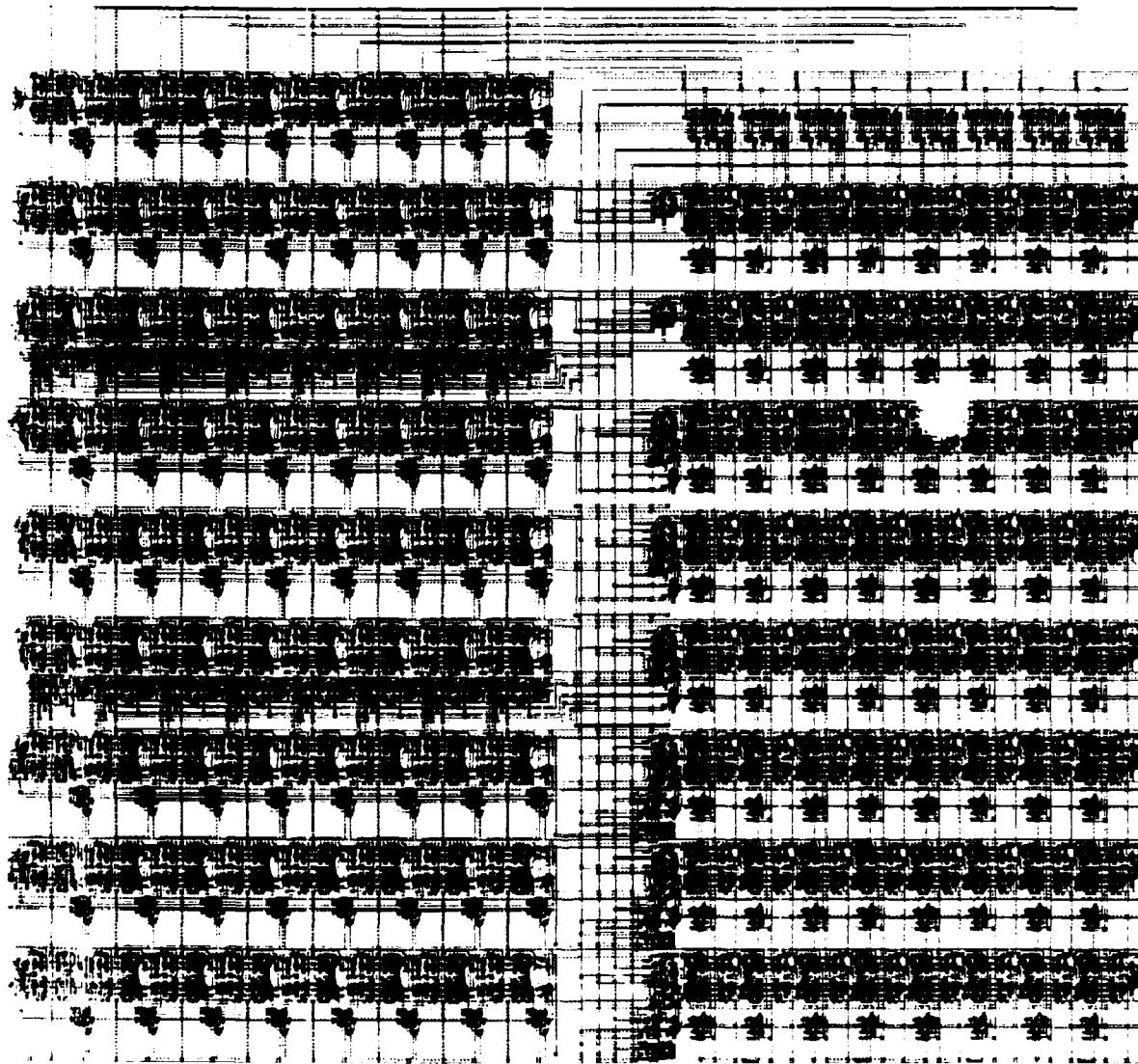


Figure 30. The $A + BC$ Cell

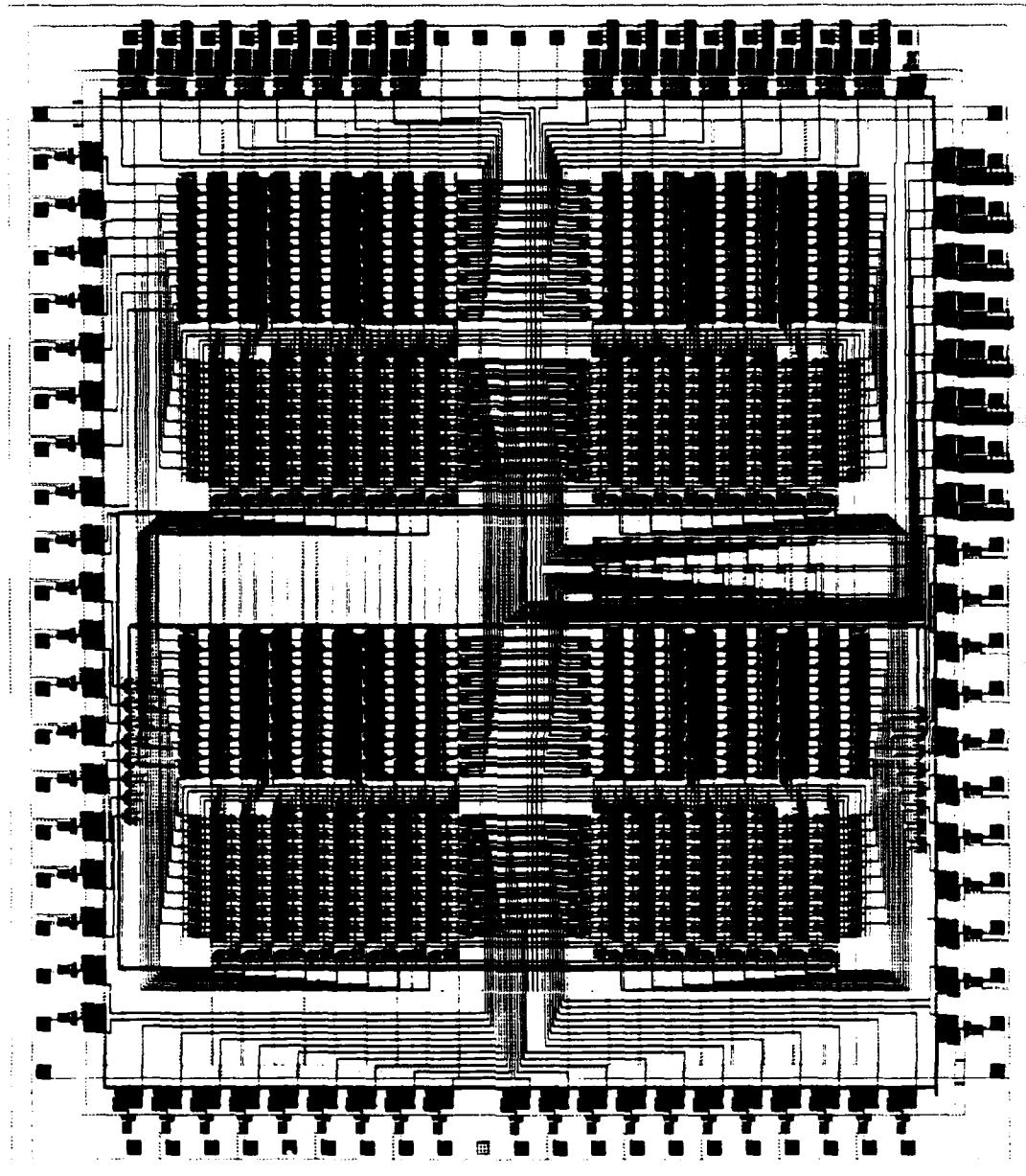


Figure 31. The LU Architecture Chip

SECTION 6

ALGEBRAIC-INTEGER-TO-ANALOG (OR DIGITAL) CONVERSION

After processing in an algebraic-integer RNS system is completed, the results are converted from the inner and outer level of RNS parallelism back into algebraic-integer form. The algebraic integers can be assumed to be of the form

$$a_0\omega_0 + a_1\omega_1 + \cdots + a_{n-1}\omega_{n-1}, \quad (6.1)$$

where the coefficients a_i are integers, which are usually much larger than at the input due to growth during the processing, and $\{\omega_0, \omega_1, \dots, \omega_{n-1}\}$ corresponds to the algebraic-integer basis. The final quantization process, referred to as *requantization* to distinguish it from the input quantization problem, involves evaluating (6.1) using finite precision approximations of the ω_i s.

This final evaluation can be real or complex depending on the choice of algebraic integers. If the ω_i s are represented in binary form, then a binary result is computed. If desired, this binary result can be input to a conventional digital-to-analog (D/A) converter to obtain an analog result.

The key issue in this section is determining how accurate the approximations of the ω_i s have to be so that requantization degrades only negligibly the overall performance. That the coefficients in (6.1) can become quite large would seem to imply that the ω_i s may have to be approximated very accurately. However, this has turned out to not always be the case. In the case of computing a fast Fourier transform (FFT) with cyclotomic integers, [4] reported that extensive simulations showed that approximations (in this case of powers of ω —a complex 16th root of unity) having comparable precision to those employed in the input quantization and for the *twiddle* factors were generally sufficient, with very little gain in performance if more accurate approximations were used. The paper [4] also provided a statistical analysis as to why this was so.

This section deals with issues common to general algebraic-integer requantization, using the algebraic integers $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$ for illustration. Performance data, which is consistent with the conclusion in [4], is presented for three FIR filter examples. An error analysis shows, for a processing function consisting of a sum of products (which are assumed to be orthogonal random variables), that the performance degradation, expressed as a difference of output SNRs, is independent of the output coefficient size and summation length. The requantization for complex algebraic integers can be reduced to the real case by treating real and imaginary parts separately. This is illustrated for $\mathbb{Z}[e^{2\pi i/16}]$ using $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$. Finally, implementation issues are discussed, including the possibility of implementing the requantization using an RNS.

6.1 ALGEBRAIC-INTEGER REQUANTIZATION PERFORMANCE

The final stage in any algebraic-integer RNS system is the evaluation of (6.1) using finite precision approximations of elements of the algebraic-integer basis. The ring $\mathbb{Z}[\sqrt{2+\sqrt{2}}]$ represented with the nonpolynomial basis $\mathbf{B} = \{1, \sqrt{2+\sqrt{2}}, \sqrt{2}, \sqrt{2-\sqrt{2}}\}$ will be used to illustrate the issues involved. The numbers in this case have the form

$$a = a_0 + a_1\sqrt{2+\sqrt{2}} + a_2\sqrt{2} + a_3\sqrt{2-\sqrt{2}}. \quad (6.2)$$

As before, $\mathbb{Z}[\mathbf{B}]_M$ denotes the set of elements of the form (6.2) with coefficients in the range $[-M/2, M/2]$.

To form the output a in this case, an inner product of the algebraic-integer coefficient vector $\mathbf{A} = (a_0, a_1, a_2, a_3)$ must be taken with the *basis vector* Ω ,

$$\Omega = (1, \sqrt{2+\sqrt{2}}, \sqrt{2}, \sqrt{2-\sqrt{2}}).$$

This task is complicated by the fact that three of the components of the basis vector cannot be represented exactly as binary fractions. The obvious solution—use as many bits as may be required for the basis vector components—may not be feasible, as more bits translates into slower evaluations. This could make this final evaluation the system bottleneck.

A practical requirement is that the components of Ω be represented by numbers of b bits or less, where b is determined by the speed requirements of the system. A degree of freedom still exists even when b has been specified, due to the fact that a scaled version of the output a is just as good as a itself as long as the scale factor remains constant. Thus, one can choose to represent $k\Omega$, k a constant scale factor, rather than Ω itself if this yields a better approximation. In either case, the b -bit approximation of the basis vector is $\hat{\Omega}$,

$$k\hat{\omega}_i = \min \{ [k\omega_i], 2^b - 1 \}, \quad i = 0, 1, 2, 3, \quad (6.3)$$

where ω_i is the i th component of Ω , and the square brackets indicate a rounding to the nearest integer (k is considered to be 1 for the non-scaled approximation). The advantages of scaling can be seen when approximating the numbers $(\sqrt{2}, \sqrt{8})$. Using the scale factor $k = 1/\sqrt{2}$ results in the numbers $(1, 2)$, which can be represented exactly with only two bits.

How does one choose the proper scale factor k given the basis vector and the number of bits used in approximating it? Assume that the rescaled (the factor of k has

been removed) exact output is a , and the rescaled output of a finite-precision evaluation is \hat{a} . Then the quantity to be minimized is the squared error $(a - \hat{a})^2$. Letting $H = \Omega - \hat{\Omega}$, the error vector due to the approximation of Ω , then

$$(a - \hat{a})^2 = (a_0\eta_0 + a_1\eta_1 + a_2\eta_2 + a_3\eta_3)^2, \quad (6.4)$$

where η_i is the i th component of H . On the assumption that the coefficients a_i are independent, zero-mean, and identically distributed random variables, then the expected error due to the basis vector approximation error is

$$\begin{aligned} E((a - \hat{a})^2) &= E(a_0^2)\eta_0^2 + E(a_1^2)\eta_1^2 + E(a_2^2)\eta_2^2 + E(a_3^2)\eta_3^2 \\ &= E(a_0^2)(\eta_0^2 + \eta_1^2 + \eta_2^2 + \eta_3^2). \end{aligned} \quad (6.5)$$

That this error depends on the expected value of the square of the output coefficient size means that the requantization error is dependent on the processing function.

The quantity in (6.5) is minimized by minimizing the sum of the squares of the basis vector approximation errors η_i . Using (6.3), (6.5), and the equation for H , it can be shown that that sum is a piecewise-continuous quadratic function of k containing a finite number of minima. These can be solved for exactly and compared to find the scale factor that minimizes the squared approximation error. Table 9 gives optimal values of k for b ranging from 8 to 12 bits. In addition, the integer approximation $k\hat{\Omega}$ (which is what the requantization hardware would actually use) and the rescaled approximation $\hat{\Omega}$ are given for the various values of b . The first four entries are the exact values of the components of Ω to six decimal places and are there for comparison.

To test the performance of finite-precision requantization, a simulation of an algebraic-integer FIR filter was designed. Three filters—a lowpass filter, a highpass filter, and a multiple passband/stopband filter—were simulated on nine sets of test input data. The test data consisted of a combination of sinusoids and additive white Gaussian noise. For each run, an algebraic-integer range M was chosen. The set $\mathbf{Z}[\mathbf{B}]_M$ was then used to quantize the input data by going through a 12-bit A/D converter and mapping to the appropriate element in $\mathbf{Z}[\mathbf{B}]_M$ (the two-stage approach of section 3.2). The quantized data was put through a filter arrived at by quantizing one of the three filters—initially designed with floating-point coefficients—with the closest scaled element of $\mathbf{Z}[\mathbf{B}]_M$. Scaling here is used in the same manner and for the same reasons as it was in approximating Ω , only now the set of available representation levels consists of algebraic integers rather than just integers as before (simulation design issues are covered in greater detail in the next section).

To measure the effect on performance of the finite-precision final evaluation, the performance when this evaluation is performed exactly must first be measured. This was done by comparing the output of the algebraic-integer simulation, assuming exact

**Table 9. Optimal Approximations for b -bit Requantization
for $\mathbb{Z}[\sqrt{2} + \sqrt{2}]$**

bits	k	$k\hat{\Omega}$	$\hat{\Omega}$
∞	—	—	1.000000 1.847759 1.414214 0.765367
8	110.983	111 205 157 85	1.000157 1.847136 1.414636 0.765886
9	255.992	256 473 362 196	1.000029 1.847711 1.414104 0.765648
10	473.016	473 874 669 362	0.999966 1.847717 1.414328 0.765302
11	1091.042	1091 2016 1543 835	0.999961 1.847775 1.414244 0.765323
12	1759.977	1760 3252 2489 1347	1.000013 1.847751 1.414223 0.765351

requantization, with the output of a floating-point simulation (i.e., a simulation where neither the floating-point input data nor the original floating-point filter coefficients were quantized, and where all computations took place in full floating-point accuracy). The performance of the algebraic-integer simulation with exact requantization was measured by computing the signal-to-noise ratio (SNR) of that simulation with respect to the floating-point simulation.

Table 10. SNR Degradation for b -bit Polynomial Evaluation

bits	ΔSNR as a function of the range M					
	2	4	6	8	10	12
8	0.01	0.20	0.84	3.63	7.09	10.41
9	0.01	0.05	0.11	1.04	1.43	3.49
10	0.00	-0.01	-0.02	0.09	0.24	1.91
11	0.00	-0.01	0.01	-0.01	-0.03	0.55
12	0.00	0.00	0.01	0.00	0.02	0.26

To be more precise, let the j th output of the floating-point simulation for a specific set of input data and a specific filter be called α_j , and let the j th output of the analogous algebraic-integer exact requantization simulation be called a_j . Then the SNR with exact requantization is

$$\text{SNR}_1 = 10 \log_{10} \left[\frac{\sum_j \alpha_j^2}{\sum_j (\alpha_j - a_j)^2} \right]. \quad (6.6)$$

Call the j th output of the finite-precision polynomial evaluation simulation \hat{a}_j . Then the SNR in that case is

$$\text{SNR}_2 = 10 \log_{10} \left[\frac{\sum_j \alpha_j^2}{\sum_j (\alpha_j - \hat{a}_j)^2} \right]. \quad (6.7)$$

The *degradation* due to using $\hat{\Omega}$ to generate \hat{a}_j rather than using Ω to generate a_j , is the difference between the two SNRs, i.e.,

$$\Delta\text{SNR} = \text{SNR}_1 - \text{SNR}_2. \quad (6.8)$$

The above quantity has been computed for two different sets of input data put through all three filters—in other words, six combinations. Each of these combinations was run using $M = 2, 4, 6, 8, 10, 12$ and requantization precisions of 8 to 12 bits, as well as an exact floating-point requantization. In addition, a floating-point simulation of each combination was run to set up a signal reference against which SNRs could be measured. The results were divided according to M and b , and are tabulated in table 10.

The advantages of scaling before approximation, as done in (6.3), can be seen in the fact that an SNR degradation of more than 13 dB was observed when scaling was not used in the $b = 8$ bits, $M = 8$ case (as compared to only a 3.63-dB degradation for an optimal scaled approximation). In that instance, the ω_i s were approximated by writing them as binary fractions and rounding to 8 bits.

It seems clear from the table that degradation in SNR for a given precision b is directly linked to the size of M , which in turn is related to the accuracy of the input and filter coefficient approximations. If the limit on acceptable degradation is set at 0.5 dB, then a requantization precision set at the same level as the input and filter coefficient precision is all that is required to obtain acceptable performance. For example, if $M = 4$, which corresponds to better than 8-bit performance (see section 3.4), $b = 8$ suffices.

The reader may wonder why some (albeit small) negative SNR degradations occur in table 10, indicating that the imprecise requantization *improved* performance, rather than degrading it as expected. The reason for this lies in the fact that the imprecise evaluation merely adds more errors onto an already corrupted result. On average, one expects the error to increase as a result, but there is no reason why the two errors being added might not tend to cancel each other in specific cases. Since only six runs are being averaged to produce each entry in table 10, it is not surprising that a noticeable variation from the expected degradation occurs. When this expected degradation is very close to zero, the variation can result in negative Δ SNRs. One would expect this phenomenon to disappear for very large sample sizes.

6.2 AN ERROR ANALYSIS

The results in table 10 apply only to the finite-impulse response filtering simulations that were run. A more valuable result would be an error analysis of requantization in a more general processing framework.

System performance for finite-impulse response filtering is usually measured by SNR. That is why SNR degradation was considered to be an appropriate measure of finite precision requantization in section 6.1. In a more general setting SNR may be an inconvenient way of quantifying errors. Even worse, it may not even be a relevant measure at all. The ultimate error measure is, of course, the errors themselves, but these are usually cumbersome to work with directly. If one assumes the errors due to requantization are a sequence of zero-mean uncorrelated random variables (a common assumption) then the most common measure of the error sequence is the expectation of the squared error. Call this σ_r^2 ,

$$\sigma_r^2 = E((a - \hat{a})^2). \quad (6.9)$$

where E indicates an expectation (the time index j is omitted in this section under the assumption that all sequences are wide-sense stationary). By (6.5), σ_r^2 is proportional to the expected square of the algebraic integer coefficient a_0 . Under the assumptions

used in (6.5), it can be shown that

$$\begin{aligned}
 E(a^2) &= E((a_0 + a_1\sqrt{2+\sqrt{2}} + a_2\sqrt{2} + a_3\sqrt{2-\sqrt{2}})^2) \\
 &= (1 + (2 + \sqrt{2}) + 2 + (2 - \sqrt{2}))E(a_0^2) \\
 &= 7E(a_0^2).
 \end{aligned} \tag{6.10}$$

Assuming a sum-of-products process, each ideal output term α is a sum of N product terms p_i ,

$$\alpha = \sum_{i=1}^N p_i,$$

and each p_i is actually approximated (because of initial quantizations) by \hat{p}_i with an error $e_i = \hat{p}_i - p_i$, so that

$$a = \sum_{i=1}^N \hat{p}_i.$$

Assuming the \hat{p}_i are identically distributed orthogonal random variables,

$$E(a^2) = E\left(\left(\sum_i \hat{p}_i\right)^2\right) = NE(\hat{p}_i^2). \tag{6.11}$$

Using (6.5), (6.10), and (6.11) in (6.9) yields

$$\sigma_r^2 = \frac{NE(\hat{p}_i^2)}{7}(\eta_0^2 + \eta_1^2 + \eta_2^2 + \eta_3^2). \tag{6.12}$$

It can be seen that the requantization error grows linearly as the number of product terms N .

Even though the main focus of section 6 is the effect of requantization, one might also be interested in errors due to initial quantizations, $\sigma_q^2 = E((\alpha - a)^2)$. It is not difficult to show that

$$\sigma_q^2 = NE(e_1^2), \tag{6.13}$$

if one assumes that the e_i are identically distributed zero-mean uncorrelated random variables. Given σ_r^2 , σ_q^2 , and the expected signal power $E(\alpha^2) = NE(p_1^2)$, one can construct almost all conceivable interesting measures of error (i.e., total error variance $\sigma_q^2 + \sigma_r^2$, relative error variances, etc.). Since $E(\hat{p}_i^2) = E(p_i^2) + E(e_i^2)$, and since one has explicit control over N and the basis vector approximation error, the three quantities specified in (6.11)–(6.13) depend only on $E(\hat{p}_i^2)$ and $E(e_i^2)$. These latter two quantities are specific to the processing situation, but once they have been determined the error

performance of the system can be completely characterized for all values of N and all basis vector approximations. Hence the goal of this section—to determine how the accuracy of the basis vector approximation effects the overall performance—can be achieved using (6.11)–(6.13) without need for experimental determination. For example, the accuracy needed in the basis vector approximation can be determined from (6.12) when $E(\hat{p}_i^2)$ has been determined, and when an acceptable amount of requantization error has been decided upon.

Applying these results to the filtering example, consider the following error measure:

$$\text{ESNR}_1 = 10 \log_{10} \left[\frac{E(\alpha^2)}{E((\alpha - a)^2)} \right]. \quad (6.14)$$

It is not unreasonable to expect SNR_1 to behave like the quantity ESNR_1 as

$$\frac{1}{J} \sum_{j=1}^J \alpha_j^2 \rightarrow E(\alpha^2) \quad \text{as } J \rightarrow \infty \quad (6.15a)$$

$$\frac{1}{J} \sum_{j=1}^J (\alpha_j - a_j)^2 \rightarrow E((\alpha - a)^2) \quad \text{as } J \rightarrow \infty \quad (6.15b)$$

where convergence occurs with probability 1.

Proceeding in a like manner with SNR_2 , its analogue is ESNR_2 ,

$$\text{ESNR}_2 = 10 \log_{10} \left[\frac{E(\alpha^2)}{E((\alpha - \hat{a})^2)} \right]. \quad (6.16)$$

Thus, the degradation in the *expected* signal to *expected* noise ratio is

$$\begin{aligned} \Delta \text{ESNR} &= \text{ESNR}_1 - \text{ESNR}_2 \\ &= 10 \log_{10} \left[\frac{E((\alpha - \hat{a})^2)}{E((\alpha - a)^2)} \right]. \end{aligned} \quad (6.17)$$

Assuming that the errors due to quantization ($\alpha - a$) are uncorrelated with the errors due to requantization ($a - \hat{a}$), it follows that

$$\begin{aligned} E((\alpha - \hat{a})^2) &= E((\alpha - a)^2 + (\alpha - \hat{a})^2 + 2(\alpha - a)(a - \hat{a})) \\ &= E((\alpha - a)^2 + (a - \hat{a})^2). \end{aligned} \quad (6.18)$$

Substituting (6.18) into (6.17),

$$\begin{aligned} \Delta \text{ESNR} &= 10 \log_{10} \left[1 + \frac{E((a - \hat{a})^2)}{E((\alpha - a)^2)} \right] \\ &= 10 \log_{10} E \left[1 + \frac{\sigma_r^2}{\sigma_q^2} \right]. \end{aligned} \quad (6.19)$$

A straightforward substitution yields

$$\Delta\text{ESNR} = 10 \log_{10} \left[1 + \frac{(\eta_0^2 + \eta_1^2 + \eta_2^2 + \eta_3^2) E(\hat{p}_1^2)}{7E(e_1^2)} \right]. \quad (6.20)$$

The felicitous and surprising result here is that (6.20) does not depend on the summation length N or $E(a_0^2)$. For a fixed processing function consisting of a sum of products (with no assumption on the number of terms in the products needed), the ΔESNR , due to imprecise requantization, will not change when the summation is lengthened, even though the size of the algebraic-integer coefficients increases due to the longer computation. This says that one can use table 10 to predict the performance of linear filters of arbitrary length.

Finally, note that the expression for ΔESNR is in qualitative agreement with the results of table 10. The size of $E(\hat{p}_1^2)$ will hardly be affected by the range M , but as M is increased, the quantity $E(e_1^2)$ will decrease, resulting, for a fixed requantization level ($\eta_0^2 + \eta_1^2 + \eta_2^2 + \eta_3^2$ is constant), in an increase in the fraction of (6.20). This agrees with the increasing values of ΔESNR along the rows of table 10.

6.3 COMPLEX ALGEBRAIC-INTEGER REQUANTIZATION

Complex algebraic-integer requantization involves a sum of products where each product, in general, involves an integer times a complex number. Rather than using a full complex multiplier, it is more efficient to treat the product as two real multiplications. Thus, it is natural to treat real and imaginary parts separately and reduce the calculation to one involving real algebraic integers. This approach is illustrated for $\mathbb{Z}[e^{2\pi i/16}]$ using the real subring $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$. In the following, $\omega = e^{2\pi i/16}$.

An element $x + yi = a_0 + a_1\omega + \dots + a_7\omega^7$ of $\mathbb{Z}[\omega]$ has real and imaginary parts given by

$$x = a_0 + \frac{a_1 - a_7}{2} \sqrt{2 + \sqrt{2}} + \frac{a_2 - a_6}{2} \sqrt{2} + \frac{a_3 - a_5}{2} \sqrt{2 - \sqrt{2}}, \quad (6.22)$$

$$y = a_4 + \frac{a_3 + a_5}{2} \sqrt{2 + \sqrt{2}} + \frac{a_2 + a_6}{2} \sqrt{2} + \frac{a_1 + a_7}{2} \sqrt{2 - \sqrt{2}}. \quad (6.23)$$

These formulas can be derived by substituting $\omega = \sqrt{2 + \sqrt{2}}/2 + i\sqrt{2 - \sqrt{2}}/2$, or by appealing to the results of section 3.3, in particular the change of basis matrix (3.13).

Formulas (6.22) and (6.23) imply that to implement the requantization for $\mathbb{Z}[\omega]$ (expressed in terms of the polynomial basis), the vector of coefficients (a_0, a_1, \dots, a_7) can be transformed by the change of basis matrix given in (3.13), and the resulting

vector split in half and input into two $\mathbb{Z}[\sqrt{2} + \sqrt{2}]$ requantizers, which are implemented using the nonpolynomial basis. The entries in the transformation (3.13) can be scaled by 2 to eliminate the fractions $\pm 1/2$, and the transformation implemented with three additions, three subtractions and two shifts (for implementing the multiplications by 2).

6.4 IMPLEMENTATION ISSUES

The final requantization involves forming an inner product between the current vector of algebraic-integer coefficients and a fixed vector of basis approximations. As such, the fundamental operation is a multiply and accumulate. Off-the-shelf components can be used to implement this function provided the speed, algebraic-integer coefficient size, and basis approximation accuracy requirements can be satisfied. Currently, commercial multiply-and-accumulate components are available that perform 16-bit \times 16-bit fixed-point multiplications with up to 40 bit accumulations with clock cycle time ranging from 50 to 100 ns (up to 20 MHz throughput rate).

If the accuracy requirements of an application imply that large algebraic-integer coefficients and highly accurate basis approximations must be used in the requantization, and if high throughput is required, then performing the requantization using an RNS may be the best solution. In this case, the basis approximations are scaled to integers using an appropriately large scale factor. The RNS for the requantization is formed by adding moduli to the the existing RNS of the algebraic-integer processor to contain the increased range. The residue representations that result from the inner-level reconversion are input directly to the second RNS processor. The inputs corresponding to the additional moduli are determined by a base extension process, one for each coordinate position. A final processor that implements the Chinese Remainder Theorem (CRT) is required . The fractional representation method of section 4 is suggested, since the full output precision is probably not needed at this stage.

Is is interesting to observe that the two-stage RNS processor described above can be viewed as a single RNS process. A vector of residues produced by the inner channels for a given moduli is multiplied by the reconversion matrix. Next, requantization is performed by computing the inner product of the resulting vector and the vector of residues of the basis approximations, with the result being input to the final CRT processor. Instead of computing the result in this order, however, the inner-level conversion and the final requantization can be combined by premultiplying the reconversion matrix by the vector of basis residues, to produce a *reconversion* vector. The output conversion then involves taking the inner product of this reconversion vector and the vector of residues produced by the inner channels, with the result still being input to the CRT processor.

The practicality of the simplified algebraic-integer RNS processor described above depends in large part on the number of extra moduli that are needed to contain the dynamic range of the requantization result. Additional processing channels (replicated by the degree of the algebraic integer-extension) are required for each of the additional moduli. Since the processing should be of sufficient complexity to amortize the overhead of RNS conversion and reconversion, even the addition of a single extra moduli may negate the advantage of the simplified output conversion. In many processing situations the range required to contain the requantization result will be considerably larger, and the two-stage RNS implementation will be more appropriate. If, however, the dynamic range after requantization is comparable to (or smaller than) the dynamic range of the original RNS, as would be the case if the requantization values correspond to very small positive quantities, then the simplified output conversion method would be advantageous.

6.5 CONCLUSION

This section treated the final requantization from the algebraic-integer number representation back to a conventional binary or analog representation. An example of an FIR filter (a sum of products) with numerical quantities represented by elements of $\mathbb{Z}[\sqrt{2} + \sqrt{2}]$ was used to illustrate the issues involved, including the desirability of scaling the basis values to improve their approximations and the efficient treatment of complex requantization. In this case, the degradation of the overall performance due to requantization errors was negligible provided the basis values were approximated with a precision comparable to the precision used for the inputs and filter coefficients. A similar conclusion was reached in [4] for the FFT. An error analysis showed that, for a general sum-of-products calculation, where the products are assumed to be orthogonal random variables, the performance degradation, expressed as a difference of SNRs, is independent of the algebraic-integer coefficient sizes and the length of the summation.

However, there is no guarantee that the heuristics derived for the FIR filter (product with two terms) or the FFT (most of the simulation results in [4] were for a 256-point radix 16 FFT—a sum of 3-term products) will apply to arbitrary processing situations or functions, particularly situations or functions that involve product terms significantly larger than considered in these past studies. The error analysis implied that the magnitude of the products themselves adversely affects the requantization error. These magnitudes depend, at least, on the magnitude of the input data and on the number of terms involved in the products. As one example of a more complicated situation, computing a single term of the adjoint of a $k \times k$ matrix requires a sum consisting of $(k - 1)!$ products with $(k - 1)$ terms each. In such situations, studies similar to the ones in this section would have to be conducted to obtain the appropriate requantization precision heuristic. The performance degradation due to finite precision requantization should be independent of the number of terms in the sum, so this

calibration of the requantization precision can be determined using sums with shorter length.

Commercial multiply-and-accumulate chips can be used to implement the sum-of-products calculation involved in the requantization. Current devices can handle coefficient sizes and basis approximations of up to 16 bits at a clock rate of up to 20 MHz. For applications with more severe accuracy or speed requirements, an RNS implementation of the requantization is suggested, yielding a two-stage RNS implementation. A simpler RNS implementation, which combines the inner-level conversion with the requantization, is possible if the increase in the dynamic range requirements due to requantization are negligible.

SECTION 7

FUTURE WORK AND CONCLUSION

The objective of this work was to provide solutions to the quantization and conversion problems involved in algebraic-integer processing, thus paving the way for a variety of algebraic-integer RNS processors. Future work related to this effort could involve the implementation of the designs/structures contained in this report as well as additional studies of the quantization properties of finite sets of algebraic integers (for example, the conjecture at the end of section 3.3 concerning the performance of the separate real and imaginary complex quantization scheme). More importantly, the utility of algebraic-integer RNS processing must be evaluated by comparing the method with other processing schemes in the case of specific applications.

This section outlines future work involving an algebraic-integer brassboard, which initially will be used to implement an algebraic-integer RNS finite-impulse-response (FIR) filter. In part, the purpose of the brassboard is to have a working algebraic-integer RNS processor for near-term demonstration purposes. A preliminary simulation analysis comparing integer and algebraic-integer processing in the case of an FIR filter, which serves to indicate the scope of the brassboard, is described. A final section summarizes the conclusions of this report, including some final thoughts on some applications that are likely to benefit the most from using the algebraic-integer number representation.

7.1 ALGEBRAIC-INTEGER BRASSBOARD

An algebraic-integer brassboard for implementing and testing the quantization and conversion functions described in this report is being planned. In addition to demonstrating that the conversion functions, when implemented successively, yield the identity function, the brassboard will be used to configure an algebraic-integer transversal filter. Surplus RNS transversal filter chips, which were fabricated under a separate effort for an experimental wideband high-frequency communication system, will be used to implement the processing function. The performance of the resulting algebraic-integer transversal filter will be evaluated.

Two different, but related, demonstrations are planned. First, a real algebraic-integer processor that uses the 4th degree extension $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$, referred to as the *real demonstration*, will be built. Two of the primes for which transversal filter chips are available, 17 and 31, are suitable for this ring. The second processor, referred to as the *complex demonstration*, will use the 8th degree cyclotomic extension $\mathbb{Z}[e^{2\pi i/16}]$. In this case, only the chips for the prime 17 can be used. Appendix B lists the inner-level conversion matrices for these rings and primes.

Since $\mathbb{Z}[\sqrt{2 + \sqrt{2}}]$ corresponds to the real numbers in $\mathbb{Z}[e^{2\pi i/16}]$, the complex demonstration can take advantage of the hardware developed for the real demonstration. The two-stage quantization approach of section 3.2 will be used in the real demonstration, and the approach in section 3.3 of treating real and imaginary parts separately will be used to reduce the complexity of the complex quantization problem. Similarly, the hardware developed for requantization in the real demonstration can be replicated and used, as in section 6.2, to perform the requantization in the complex demonstration.

In the two-stage quantization approach of section 3.3, tables are required to store the algebraic-integer approximations. These approximations can be converted ahead of time into the inner level of RNS parallelism and be stored in the table in this form, eliminating the need to explicitly implement the conversion into the two levels of RNS parallelism. Of course, this requires a table for each prime in the RNS. The conversion out of the inner level of parallelism will be implemented with the chip described in section 5. The real demonstration will require a 2×2 grid of these chips for each modulus; the complex demonstration will require a 4×4 grid.

The conversion in the brassboard out of the outer level of RNS parallelism, which involves integer RNS, will either be performed using a table-based converter (in the case of the real demonstration with the moduli 17 and 31) or not be required at all (in the case of the complex demonstration with only the modulus 17). This approach is consistent with the fact that the reduced ranges involved in algebraic-integer RNS decrease the complexity of the integer-RNS outer level reconversion, making the feasibility of table-based converters more likely. The final requantization will be implemented using commercial multiply-and-accumulate chips.

7.2 PERFORMANCE OF AN ALGEBRAIC-INTEGER FIR FILTER

This section reports on the results of a simulation activity that was undertaken to determine if the proposed algebraic-integer-RNS FIR filter planned for the brassboard could provide a meaningful demonstration. The proposed implementation will be limited to the primes 17 and 31 with a dynamic range (per coefficient) of $17 \times 31 = 527$; i.e. the absolute value of the coefficients at the output must be less than or equal to 263. Three different filters were considered in this preliminary study. The actual filters to be implemented will be subjected to a similar, but more exhaustive, simulation study. The present effort is an initial attempt at comparing integer and algebraic-integer processing.

A simulation of an FIR filter was designed in order to (1) evaluate the error vs. range performance of the algebraic integers $\mathbb{Z}[B]_M$, for

$$B = \{1, \sqrt{2 + \sqrt{2}}, \sqrt{2}, \sqrt{2 - \sqrt{2}}\},$$

and (2) compare their performance to that of the integers. The simulation consisted of three fundamental blocks: the initial quantization of the input data (to either integers or algebraic integers); the filtering operation, where the quantized data was convolved with the quantized filter coefficients; and the final requantization for the algebraic-integer case. In this application, finite-precision requantization can be performed with essentially no loss in performance without having to resort to a second RNS processor. Thus, in this preliminary study, the requantization is performed using floating-point numbers, corresponding to it being performed by commercial multiply-and-accumulate chips of more than adequate accuracy. The primary purpose of this study is to determine the range requirements of the respective RNS processors for equivalent levels of performance.

The first task in setting up a simulation of this type is the choice and design of a set of filters whose coefficients are drawn from either \mathbb{Z}_M , indicating the integers modulo M , or $\mathbb{Z}[\mathbf{B}]_M$. Unfortunately, finite-wordlength filter design is an open problem at present. More specifically, for most filter design criteria a deterministic way of finding the best coefficients drawn from the available set is not known, short of exhaustive search. This is mainly due to the difficulty of fitting the discrete, finite-wordlength constraints into the analytic methods employed in filter design.

The current practice in finite-wordlength filter design is to design an infinite-wordlength filter, then approximate it in some manner with finite-wordlength representatives. The unexpected problems that occur in this process can be seen in the following example. A lowpass filter was designed and approximated by two finite-wordlength sets— $\mathbb{Z}[\mathbf{B}]_{10}$ and $\mathbb{Z}[\mathbf{B}]_{12}$. The approximations were chosen so as to minimize the squared error of the finite-wordlength filter coefficients. Naturally, the filter designed using $M = 12$ had a lower squared error than that using $M = 10$. However, on a set of test input data consisting of two sinusoids, one in the passband and one in the stopband, plus some additive white Gaussian noise (AWGN), the $M = 10$ filter performed better than the $M = 12$ filter even though the input data were quantized less accurately in the $M = 10$ case. This occurred, even though its total squared error was higher, because the $M = 10$ filter preserved a frequency notch better than the other filter and one of the sinusoids happened to fall close to that notch. It is important to remember that squared error is not always the best approximation performance measure, and that a filter approximation that is good in one particular situation may not be suitable in another.

For the simulation under discussion, the filters were designed by first designing an infinite-wordlength filter of the desired type and then approximating the infinite-wordlength coefficients with the closest scaled representative in \mathbb{Z}_M or $\mathbb{Z}[\mathbf{B}]_M$. There were three infinite-wordlength filters designed—a 55-tap filter with three stopbands and two passbands, a 60-tap lowpass filter, and a 61-tap highpass filter. These fil-

ters were approximated by $Z[B]_M$ for $M = 2, 4, 6, 8, 10, 12$, and by Z_M for $M = 2^6, 2^7, 2^8, 2^9, 2^{10}, 2^{11}, 2^{12}$.

The input data fed into the filters consisted of a combination of sinusoids with AWGN. There were ten such data sets computed, each 150 samples long and consisting of a combination of one or more of the frequencies 0.025, 0.125, 0.2, 0.33, and 0.45. These frequencies were chosen because none of them lay in the *don't care* bands of any of the three basic types of filters. The 55-tap multiband filter was run with all ten sets of data, while the other two were run on only two of the sets (the same two in each case).

Each run consisted of a particular set of input data quantized either by Z_M or $Z[B]_M$ (using the two-stage conversion method of section 3.2). These inputs were then convolved with a filter approximated by the same set. Along with integer and algebraic-integer processing, a baseline was established for each input-data/filter combination by filtering the floating-point input data directly with the original infinite-wordlength filter. The outputs of this process were used as the standard against which errors occurring in the integer or algebraic-integer processing could be measured.

The error was measured in terms of signal-to-noise ratio (SNR) in the same manner as in equation (6.6), i.e.,

$$\text{SNR} = 10 \log_{10} \left[\frac{\sum_j \alpha_j^2}{\sum_j (\alpha_j - a_j)^2} \right], \quad (7.1)$$

where α_j is the j th output of the floating-point run, and a_j is the j th output of the integer or algebraic-integer process. The other statistic one is interested in is the dynamic range required to contain the results. Ideally, one would like to minimize the dynamic range while maximizing the SNR. The trade-off between these two statistics is given in tables 11 and 12 for one particular set of input data run on all the filter approximations. Table 11 is for algebraic-integer processing, while table 12 is for integers. The input data set consisted of a unity-amplitude sinusoid with frequency 0.2, a unity-amplitude sinusoid with frequency 0.33, and AWGN with power 0.01. The algebraic-integer range shown was the absolute value of the largest coefficient output that occurred, while the integer range was the absolute value of the largest integer output that occurred.

The results in tables 11 and 12 should only be compared for $M = 2, 4, 6, 8$, since for larger values of M the front-end 12-bit A/D converter in the two-stage quantization method becomes a limiting factor on the accuracy of the algebraic-integer representations of the input data (see section 3.4). Although the fact that the filter coefficients are being approximated directly does result in some performance gain over the 12-bit integer case for $M = 10, 12$.

Consider $M = 4$ for example. The performance of filter #1, 48.15 dB, corresponds to a performance for the integer filter of between 8 and 9 bits; filter #2 is slightly below

Table 11. Range vs. SNR for Algebraic Integers

M	Filter #1		Filter #2		Filter #3	
	Range	SNR	Range	SNR	Range	SNR
2	42	29.86	30	22.82	53	27.93
4	147	48.15	113	41.50	106	43.60
6	187	52.80	221	54.42	236	54.46
8	386	63.68	498	63.57	383	59.80
10	526	67.90	672	66.16	583	66.97
12	799	69.53	853	68.94	830	69.55

Table 12. Range vs. SNR for Integers

bits	Filter #1		Filter #2		Filter #3	
	Range	SNR	Range	SNR	Range	SNR
6	1172	22.30	986	28.29	1046	22.49
7	5115	32.21	3937	38.65	3867	36.45
8	20144	44.34	15871	42.96	15642	40.62
9	81343	50.18	64135	47.32	62863	50.89
10	324876	53.59	255207	49.86	251371	54.83
11	1301579	61.26	1025157	59.58	1006421	60.39
12	5103488	63.43	4221821	62.21	3917674	61.29

8-bit performance; and filter #3 is between 8 and 9 bits. These levels are consistent with the accuracy of 8.53 bits for the quantizer $Z[B]_4$ against a uniform input (table 4 of section 3.4), although the present input is most certainly not uniformly distributed.

The equivalent integer filters for $M = 2, 6$, and 8 are also consistent with the accuracies of the respective algebraic-integer quantizers $Z[B]_M$. Some anomalies are to be expected, due to the problem, mentioned earlier, with using SNR as a performance measure. In summary, the results in tables 11 and 12 do not present any surprises, and it will be assumed that the integer equivalent of an algebraic-integer FIR filter can be obtained by a bit level that approximates the inputs and filter coefficients to the same level of accuracy.

An equally important point of comparison is the dynamic ranges required by the two processing methods. To get an idea of the largest range one might be called upon

to handle, the absolute values of the largest range encountered over all the runs using the 55-tap filter were determined for each M . The other two filters were not included so that filter length would be a constant factor in the measured ranges. The results for integers and algebraic integers are shown below in table 13. It should be mentioned that the dynamic range for the algebraic integer $M = 12$ case (1312) was an extreme outlier—the next largest output coefficient encountered in that or any other run was 974.

Table 13. Summary of Largest Ranges

Algebraic Integers		Integers	
M	Range	bits	Range
2	48	6	2008
4	164	7	8532
6	239	8	34896
8	524	9	138586
10	705	10	559369
12	1312	11	2234533
—	—	12	8946667

The ranges in table 13 suggest that the brassboard, with the primes 17 and 31 corresponding to an absolute value range of 263, could implement the 55-tap filter for $M = 6$. A corresponding integer filter involves approximations of between 10 and 11.42 bits (for uniform and Gaussian inputs; see tables 4 and 8) and requires a range of at least 559,369. If 5-bit (or less) primes are to be used, then the integer processor requires at least 5 moduli in the RNS ($(31 \times 29 \times 23 \times 19 - 1)/2 = 196,431$; $(31 \times 29 \times 23 \times 19 \times 17 - 1)/2 = 3,339,335$). For Gaussian inputs, this RNS may not even suffice since the range would be closer to 5,000,000.

It should be noted that the algebraic-integer system, although only requiring 2 moduli in the RNS, has 4 processing channels per moduli for a total of 8. This inefficiency, 8 channels versus 5 or 6, may be due to the fact that each coefficient of the algebraic-integer number representation experiences its own additive growth, which in this example is between 5 and 6 bits. This effect would worsen for longer filters. However, a 5- or 6-moduli RNS output converter should be much more complex than a 2-moduli RNS output converter replicated 4 times (for each of the algebraic-integer coefficients).

Finally, only the nonpolynomial basis \mathbf{B} was used in the simulations because the polynomial basis resulted in larger ranges. For example, a run with the polynomial basis that yielded a range of 72 for $M = 2$ compared to a range of 48 for the nonpolynomial basis under the same conditions.

7.3 CONCLUSION

This paper addressed the quantization and conversion problems involved in algebraic-integer RNS processing. The algebraic-integer number representation was introduced using four important examples: cyclotomic extensions involving 8th and 16th roots of unity, and their respective real subrings. A primary concern when choosing a basis for the algebraic-integer representation is to minimize the growth in coefficient size due to multiplication. This growth depends on the size of the entries in the coefficient forms, which are matrices that describe the relationships that exist between the basis elements. These relationships depend, in part, on the minimum polynomial involved in the algebraic-integer extension. In the 4th degree real example considered, this meant processing with a nonpolynomial basis. The complexity of input and output phases of a complex implementation (I and Q channels) can be reduced by representing the complex algebraic integers in terms of a product basis, although it is best, from a dynamic range viewpoint, to process using the complex polynomial basis.

Algebraic-integer RNS was introduced as a generalization of QRNS for the Gaussian integers. In algebraic-integer RNS, each parallel channel of integer RNS (the outer level of parallelism) is split further into another level of parallel channels (the inner level of parallelism). This further splitting, for a modulus m , makes the complicated algebraic-integer product easy to compute in parallel, but is only possible when the minimum polynomial of the algebraic-integer extension, considered modulo m , factors completely into distinct linear terms.

Unlike the Gaussian integers, which form the quantizer used in QRNS and correspond to a two-dimensional integer lattice, the algebraic integers form a dense representation and are a rich source of quantization problems. In practice, algebraic-integer quantizers are derived by bounding the size of the algebraic-integer coefficients. Two strategies for direct analog-to-(real)algebraic-integer conversion were described, one based on the compressor characteristic of the nonuniform algebraic-integer quantizer, and the other based on a generalization of a successive-approximation converter. A suboptimal and straightforward two-stage method of real quantization was also proposed.

In the important case that was considered, the suboptimal method performed equivalently to the direct method as long as the conventional A/D converter, which forms the front end of the two-stage method, has one bit, more conservatively two bits, of precision beyond the precision of the direct algebraic-integer quantizer. Thus, further

hardware development of a direct analog-to-algebraic-integer converter is only required for applications requiring processing at the speed and accuracy limits of current A/D technology. By treating real and imaginary parts separately, complex quantization was reduced to the real case, with a performance loss conjectured to be just a doubling in the range of the coefficients.

The algebraic-integer quantizers yield a distributed representation—a single large representative is replaced by a vector of smaller representatives. For the same level of accuracy, the total cost (measured by the number of points) of the conventional and algebraic-integer quantizers are comparable, especially when issues of robustness are considered. Thus, in the case considered—a degree 4 real extension—for the same level of accuracy, a single representative with a range R is replaced by a vector of 4 coefficients, each with a range of about $\sqrt[4]{R}$. This dramatic reduction in range at the input implies a smaller output range, hence a smaller RNS can be used, although the advantage is somewhat mitigated by the additional coefficient size growth due to the algebraic-integer multiplication. The smaller RNS range reduces the complexity of the outer level integer-RNS conversion.

Because the algebraic-integer coefficients at the beginning of the calculation are likely to be small, the conversion into the outer channels will usually be straightforward. Conversion out of these outer channels will require integer-RNS output conversion. The fractional representation method performs this output conversion without the large modulo reduction required by the Chinese Remainder Theorem. The method was modified slightly to get rid of all the modular operations at the cost of needing slightly higher precision approximations of the fractional quantities involved in the method. As a result, an implementation with conventional arithmetic is now possible, making programming to various RNSs easy. A comparison with mixed-radix conversion was made for a particular RNS, and it was shown that the active VLSI area of a fractional representation converter was always less than that of an equivalent mixed-radix one. The VLSI implementation of this converter is being developed under a separate effort.

The conversion into and out of the inner level of parallel channels, for a modulus m , is a modulo m vector-matrix multiplication. This involves a sum of products modulo m , and so is similar to the functions most likely to be implemented by the RNS processing channels; i.e., the complexities are similar also. An architecture based on the LU decomposition of the transform matrix was developed that performs this product in parallel. The LU architecture consists of a grid of basic cells and is semisystolic—the cells in each row must communicate. A chip consisting of a 2×2 grid of basic cells was designed in scalable CMOS. Each basic cell performs a multiply and accumulate modulo a programmable modulus. The modulus, and hence all values on the chip, are limited to 8 bits. The design can be clocked to 3 MHz and is currently undergoing testing before fabrication (in 3 μm technology). A degree n conversion (either into or out of the inner level) will require an $n/2 \times n/2$ grid of these chips for each modulus.

After RNS processing is completed, including the conversions out of the two levels of parallelism, the answer is obtained in algebraic-integer form. To obtain a binary or analog result, this algebraic integer must be evaluated using finite-precision approximations of the basis elements. Rather than straightforwardly rounding the basis elements to some prescribed number of bits, scaling and rounding should be employed to minimize the average squared error of these approximations. Since the output coefficients are usually quite large, there is some concern that the basis elements may have to be approximated with equally extreme precision. In the applications considered to date, this has not been the case—basis approximations with a precision matching the precision of the inputs are all that have been required for there to be no significant performance degradation due to requantization. There is no guarantee that this heuristic will apply in general.

An error analysis showed that for a processing function consisting of a sum of products, where the products were assumed to be orthogonal random variables, the performance degradation due to requantization, measured in terms of the decrease in output SNR, is independent of the output algebraic-integer coefficient size and the length of the summation. However, the magnitudes of the products do adversely affect the requantization error, and so a significantly different processing situation has to be treated on an individual basis.

The requantization can be performed with commercial multiply-and-accumulate units or even a separate RNS processor if accuracy and/or throughput requirements dictate. In the latter case, the algebraic-integer output conversion can be simplified in the special case that the dynamic range growth due to requantization is negligible. This would be the case if the final results correspond to very small positive numbers.

In the future, an algebraic-integer brassboard is planned to test the quantizers and converters that have been described in this report. Current plans also include implementing an algebraic-integer RNS FIR filter in the brassboard. A processor based on a 4th degree real extension will utilize existing processor hardware for the primes 17 and 31. A related processor for I and Q processing will use an 8th degree complex extension and the prime 17. A preliminary simulation study showed that, for a 55-tap filter, to obtain performance equivalent to the real algebraic-integer RNS processor with moduli {17, 31}, an integer RNS processor would require the 5-bit prime moduli {17, 19, 23, 29, 31}. A sixth modulus may be required depending on the distribution of the inputs and filter coefficients.

The algebraic-integer FIR filter implementation is not intended to address the fundamental question concerning the ultimate utility of the algebraic-integer number representation. Rather, it is a near-term demonstration tool, which can be completed using existing processing chips, to be used to compare the integers and algebraic integers in a situation in which we have considerable prior integer-RNS experience.

What application should be used to demonstrate the advantages of the algebraic-integer number representation? It is expected that the application will have more than one of the following characteristics: (1) sum(s) of products function (products not necessarily limited to two terms), (2) high sensitivity to quantization and round-off errors, (3) matching nonuniform input distributions, (4) high dynamic range requirement, and (5) high throughput (and low latency) requirement. There are relationships between these characteristics. For example, any modest increase in quantization performance due to (3) will translate into considerable output performance improvement because of (2). Also, difficult to quantify characteristics such as ease of design or the need for fault tolerance may enter the decision.

Most importantly, though, throughput and latency requirements probably will imply that some type of parallel processing is required. Algebraic integers, as a parallel processing scheme, are not likely to be preferred on area considerations alone. There is an inherent inefficiency with the algebraic-integer number representation (for that matter with any distributed number representation): additive growth occurs in each coefficient. An application is needed with speed and accuracy requirements that force integer processing, and dynamic ranges so large that integer RNS is impractical. Then the dynamic range relief offered by the distributed algebraic-integer representation may be essential in obtaining a practical implementation. High-performance adaptive recursive filtering applications or various types of matrix processing, where the matrices involved can have high condition numbers, could be the source of an appropriate application.

APPENDIX A

LU DECOMPOSITION

The purpose of the section is to present some results about the LU decomposition of a matrix that are needed in this paper. Let A be a nonsingular $n \times n$ matrix over some field. Then A is *LU-decomposable* if there exists a lower-triangular matrix L and an upper-triangular matrix U such that $A = LU$. Necessarily, the diagonals of L and U are nonzero; no other restriction is made on the diagonals. The requirement that A be nonsingular is not necessary for some of the results that follow; this requirement is added to ease the exposition.

An LU decomposition for a matrix is desirable for the added information that it furnishes and the simplification that it can give to calculations. For example, the LU architecture in section 5.2 uses the LU decomposition of a matrix to perform matrix multiplication. For a reference on LU decomposition, see, for example, [20].

Almost all of the results in this section are well-known (see [20]); the exception is Theorem A.2, for which no reference has been found as yet. The characterization of when a matrix is LU-decomposable is given in the following theorem. The leading $i \times i$ principal minor of a matrix M is denoted by M_i .

THEOREM A.1. *A nonsingular $n \times n$ matrix A is LU-decomposable iff the matrices A_1, A_2, \dots, A_{n-1} are all nonsingular.*

PROOF: Suppose that A is LU-decomposable and let $A = LU$ be the resulting decomposition. For $i = 1, 2, \dots, n-1$, block arithmetic yields $A_i = L_i U_i$. Since L_i and U_i are clearly nonsingular, A_i must also be nonsingular.

Conversely, let A_1, A_2, \dots, A_{n-1} be nonsingular. By induction, A_{n-1} is LU-decomposable. Let $A_{n-1} = L'U'$ be the decomposition. Then A can be factored as

$$\begin{aligned}
 A &= \begin{pmatrix} & & a_{1n} \\ & L'U' & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \\
 &= \begin{pmatrix} L' & \\ & 1 \end{pmatrix} \begin{pmatrix} 1 & & b_{1n} \\ & \ddots & \vdots \\ b_{n1} & \dots & b_{nn} \end{pmatrix} \begin{pmatrix} U' & \\ & 1 \end{pmatrix}, \tag{A.1}
 \end{aligned}$$

where $b_{n1}, \dots, b_{nn}, b_{1n}, \dots, b_{1,n-1}$ are determined from L' , U' , and the a_{ij} , and blanks denote 0s. Notice that in the last product, the first matrix is lower-triangular and the

third is upper-triangular. Since the matrix product

$$\begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ -b_{n1} & \cdots & -b_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} 1 & & b_{1n} \\ & \ddots & \vdots \\ b_{n1} & \cdots & 1 & b_{nn} \end{pmatrix}$$

is upper-triangular, the middle matrix in (A.1) is LU-decomposable. Therefore, A is LU-decomposable. ■

Implicit in the above proof is one method for calculating an LU decomposition: given A , one performs Gaussian elimination on the rows and columns to reduce A to an identity matrix, keeping track of the elementary row and column operations used. These operations can be expressed as a lower-triangular matrix and an upper-triangular matrix. The inverses of these matrices give the LU decomposition.

As an example, consider a Vandermonde matrix V (see section 5.1). Since a principal minor of a Vandermonde matrix is itself a Vandermonde matrix, Theorems 2.1 and 6.1 and induction imply that a Vandermonde matrix is LU-decomposable. This was shown by constructive methods in section 5.1.

The following theorem shows that any nonsingular $n \times n$ matrix can be made LU-decomposable by multiplication by an appropriate permutation matrix.

THEOREM A.2. *Let A be a nonsingular $n \times n$ matrix. Then there exists a permutation matrix P such that PA is LU-decomposable.*

PROOF: The proof is by induction. For $n = 2$, let

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}.$$

Then either a_{11} or a_{21} is nonzero, since A is nonsingular. Permute the rows, if necessary, to put a nonzero element in the "11" spot. The resulting matrix is LU-decomposable by Theorem A.1.

Let A be a nonsingular $n \times n$ matrix and suppose that the result is known for all nonsingular $(n-1) \times (n-1)$ matrices. Form all $(n-1) \times (n-1)$ minors of A with columns from the first $n-1$ columns. Then one of these minors is nonsingular, otherwise calculating the determinant of A by expanding along the last column would imply that A is singular. Hence, the rows of A can be permuted so that the $(n-1) \times (n-1)$ principal minor A_{n-1} is nonsingular. That is, for some permutation matrix P ,

$$PA = \begin{pmatrix} A_{n-1} & * \\ * & * \end{pmatrix}, \quad A_{n-1} \text{ nonsingular.}$$

By the induction hypothesis, there is an $(n - 1) \times (n - 1)$ permutation matrix P' such that $P'A_{n-1}$ is LU-decomposable. In particular, all of the principal minors of $P'A_{n-1}$ are nonsingular. Then so are all of the principal minors of

$$\begin{pmatrix} P' & 0 \\ 0 & 1 \end{pmatrix} PA.$$

Applying Theorem A.1 finishes the result. ■

Suppose that a nonsingular matrix A is LU-decomposable with decomposition $A = LU$. If D is any nonsingular diagonal matrix, then

$$LU = LDD^{-1}U = (LD)(D^{-1}U).$$

Hence, there is another LU decomposition of A given by LD and $D^{-1}U$. The next theorem asserts that all LU decompositions of A are of this form.

THEOREM A.3. *Let A be a nonsingular LU-decomposable matrix with decompositions LU and $L'U'$. Then, for some nonsingular diagonal matrix D ,*

$$L = L'D \quad \text{and} \quad U = D^{-1}U'.$$

PROOF: Since $A = LU = L'U'$ and everything is invertible, $(L')^{-1}L = U'U^{-1}$. The left side is a lower triangular matrix and the right side is an upper-triangular matrix, so the expression is some (nonsingular) diagonal matrix. ■

Theorem A.3 says that by forcing L and U to have some prescribed diagonals, say all 1s, then A can be written uniquely as $A = LDU$, for some diagonal matrix D . For the purposes of this paper, it is advantageous to not so restrict L and U and to in fact allow the diagonals to be determined as the context requires.

APPENDIX B

SOME ALGEBRAIC INTEGERS

In this appendix, the matrices and other information needed to perform the inner-level conversion are given for those rings of algebraic integers and primes that are likely to be used in some near-term implementations. Each ring of algebraic integers is of the form $\mathbb{Z}[\beta]$ modulo some prime p . Algebraic integers represented with the standard basis are of the form

$$a_0 + a_1\beta + \cdots + a_{d-1}\beta^{d-1}, \quad (B.1)$$

where the a_i are modulo p , and d is the degree of the extension. The minimum polynomial of β and its roots modulo p are also given; the order of the roots is consistent with the matrices.

If more than one basis is considered, each is represented as $\{\beta_0, \dots, \beta_{d-1}\}$, so that the algebraic integers are of the form

$$a_0\beta_0 + a_1\beta_1 + \cdots + a_{d-1}\beta_{d-1}.$$

The change-of-basis matrices to and from the standard representation of (B.1) is given. The entries of the change of basis matrices are rational numbers and are in fact integers whenever the two bases are both integral bases. The other matrices are consistent with the order of the basis and have entries from \mathbb{Z}_p .

The conversion, respectively reconversion, matrices are denoted by V and V^{-1} (though these matrices need not be Vandermonde matrices). The matrices in the LU architecture are L_C^{-1} and U_C for conversion and L_R^{-1} and U_R for reconversion.

Each of the R_j below is a ring of algebraic integers that will be described in this section. Each of the associated B_{jk} is a basis which generates the algebraic-integer approximations taken from R_j . It is usually the case, but not always, that each B_{jk} generates all of R_j (and so is an integral basis of R_j); any exceptions are noted. The standard basis is always B_{j1} . The order of the R_j is the order that the information is presented in this section. To avoid confusion, i here always denotes $\sqrt{-1}$:

$$R_1 = \mathbb{Z}[\sqrt{2 + \sqrt{2}}], \quad p = 17$$

$$B_{11} = \left\{ 1, \sqrt{2 + \sqrt{2}}, (\sqrt{2 + \sqrt{2}})^2, (\sqrt{2 + \sqrt{2}})^3 \right\}$$

$$B_{12} = \left\{ 1, \sqrt{2 + \sqrt{2}}, \sqrt{2}, \sqrt{2 - \sqrt{2}} \right\}$$

$$R_2 = \mathbb{Z}[\sqrt{2 + \sqrt{2}}], \quad p = 31$$

$$B_{21} = \left\{ 1, \sqrt{2 + \sqrt{2}}, (\sqrt{2 + \sqrt{2}})^2, (\sqrt{2 + \sqrt{2}})^3 \right\}$$

$$B_{22} = \left\{ 1, \sqrt{2 + \sqrt{2}}, \sqrt{2}, \sqrt{2 - \sqrt{2}} \right\}$$

$$R_3 = \mathbb{Z}[\omega], \quad \omega \text{ primitive eighth root of unity}, \quad p = 17$$

$$B_{31} = \{1, \omega, \omega^2, \omega^3\}$$

$$R_4 = \mathbb{Z}[\omega], \quad \omega \text{ primitive sixteenth root of unity}, \quad p = 17$$

$$B_{41} = \{1, \omega, \omega^2, \omega^3, \omega^4, \omega^5, \omega^6, \omega^7\}$$

$$B_{42} = \left\{ 1, \sqrt{2 + \sqrt{2}}, \sqrt{2}, \sqrt{2 - \sqrt{2}}, i, \sqrt{2 + \sqrt{2}i}, \sqrt{2}i, \sqrt{2 - \sqrt{2}i} \right\}.$$

EXAMPLE 1: Ring R_1

The ring is

$$\mathbb{Z}[\sqrt{2+\sqrt{2}}] \quad \text{modulo 17.}$$

The minimum polynomial of $\sqrt{2+\sqrt{2}}$ is

$$f(x) = x^4 - 4x^2 + 2.$$

The roots of $f(x)$ modulo 17 are

$$5, 8, 9, 12.$$

Ring R_1 with basis B_{11}

The basis is

$$\left\{ 1, \sqrt{2+\sqrt{2}}, (\sqrt{2+\sqrt{2}})^2, (\sqrt{2+\sqrt{2}})^3 \right\}.$$

Conversion Matrices:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 5 & 8 & 9 & 12 \\ 8 & 13 & 13 & 8 \\ 6 & 2 & 15 & 11 \end{pmatrix} \quad V^{-1} = \begin{pmatrix} 3 & 4 & 5 & 1 \\ 6 & 5 & 12 & 10 \\ 6 & 12 & 12 & 7 \\ 3 & 13 & 5 & 16 \end{pmatrix}$$

$$L_C^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 12 & 1 & 0 & 0 \\ 6 & 4 & 1 & 0 \\ 14 & 4 & 12 & 1 \end{pmatrix} \quad U_C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 3 & 4 & 7 \\ 0 & 0 & 4 & 11 \\ 0 & 0 & 0 & 16 \end{pmatrix}$$

$$L_R^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 4 & 6 & 0 & 0 \\ 10 & 1 & 13 & 0 \\ 3 & 13 & 5 & 16 \end{pmatrix} \quad U_R = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 7 & 8 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ring R_1 with basis B_{12}

The basis is

$$\left\{ 1, \sqrt{2+\sqrt{2}}, \sqrt{2}, \sqrt{2-\sqrt{2}} \right\}.$$

Change of Basis Matrices:

To Standard Basis

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & -3 & 0 & 1 \end{pmatrix}$$

From Standard Basis

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{pmatrix}$$

Conversion Matrices:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 5 & 8 & 9 & 12 \\ 6 & 11 & 11 & 6 \\ 8 & 12 & 5 & 9 \end{pmatrix}$$

$$V^{-1} = \begin{pmatrix} 13 & 7 & 5 & 1 \\ 13 & 1 & 12 & 10 \\ 13 & 16 & 12 & 7 \\ 13 & 10 & 5 & 16 \end{pmatrix}$$

$$L_C^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 12 & 1 & 0 & 0 \\ 8 & 4 & 1 & 0 \\ 4 & 7 & 12 & 1 \end{pmatrix}$$

$$U_C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 3 & 4 & 7 \\ 0 & 0 & 4 & 11 \\ 0 & 0 & 0 & 16 \end{pmatrix}$$

$$L_R^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 4 & 6 & 0 & 0 \\ 2 & 1 & 13 & 0 \\ 13 & 10 & 5 & 16 \end{pmatrix}$$

$$U_R = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 7 & 8 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

EXAMPLE 2: Ring R_2

The ring is

$$\mathbb{Z}[\sqrt{2 + \sqrt{2}}] \quad \text{modulo 31.}$$

The minimum polynomial of $\sqrt{2 + \sqrt{2}}$ is

$$f(x) = x^4 - 4x^2 + 2.$$

The roots of $f(x)$ modulo 31 are

$$5, 14, 17, 26.$$

Ring R_2 with basis B_{21}

The basis is

$$\left\{ 1, \sqrt{2 + \sqrt{2}}, (\sqrt{2 + \sqrt{2}})^2, (\sqrt{2 + \sqrt{2}})^3 \right\}.$$

Conversion Matrices:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 5 & 14 & 17 & 26 \\ 25 & 10 & 10 & 25 \\ 1 & 16 & 15 & 30 \end{pmatrix} \quad V^{-1} = \begin{pmatrix} 10 & 2 & 30 & 6 \\ 6 & 27 & 1 & 20 \\ 6 & 4 & 1 & 11 \\ 10 & 29 & 30 & 25 \end{pmatrix}$$

$$L_C^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 26 & 1 & 0 & 0 \\ 8 & 12 & 1 & 0 \\ 19 & 21 & 26 & 1 \end{pmatrix} \quad U_C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 9 & 12 & 21 \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

$$L_R^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 27 & 7 & 0 & 0 \\ 14 & 21 & 25 & 0 \\ 10 & 29 & 30 & 25 \end{pmatrix} \quad U_R = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 22 & 23 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ring R_2 with basis B_{22}

The basis is

$$\left\{ 1, \sqrt{2+\sqrt{2}}, \sqrt{2}, \sqrt{2-\sqrt{2}} \right\}.$$

Change of Basis Matrices:

$$\begin{array}{c} \text{To Standard Basis} \\ \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & -3 & 0 & 1 \end{array} \right) \end{array} \quad \begin{array}{c} \text{From Standard Basis} \\ \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{array} \right) \end{array}$$

Conversion Matrices:

$$V = \left(\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 5 & 14 & 17 & 26 \\ 23 & 8 & 8 & 23 \\ 17 & 5 & 26 & 14 \end{array} \right) \quad V^{-1} = \left(\begin{array}{cccc} 8 & 20 & 30 & 6 \\ 8 & 25 & 1 & 20 \\ 8 & 6 & 1 & 11 \\ 8 & 11 & 30 & 25 \end{array} \right)$$

$$L_C^{-1} = \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 26 & 1 & 0 & 0 \\ 10 & 12 & 1 & 0 \\ 9 & 24 & 26 & 1 \end{array} \right) \quad U_C = \left(\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 0 & 9 & 12 & 21 \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 0 & 5 \end{array} \right)$$

$$L_R^{-1} = \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 27 & 7 & 0 & 0 \\ 2 & 21 & 25 & 0 \\ 8 & 11 & 30 & 25 \end{array} \right) \quad U_R = \left(\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 0 & 1 & 22 & 23 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{array} \right)$$

EXAMPLE 3: Ring R_3

The ring is

$\mathbb{Z}[\omega]$, ω primitive eighth root of unity, modulo 17.

The minimum polynomial of ω is

$$f(x) = x^4 + 1.$$

The roots of $f(x)$ modulo 17 are

$$2, 8, 15, 9.$$

Ring R_3 with basis B_{31}

The basis is

$$\{1, \omega, \omega^2, \omega^3\}.$$

Conversion Matrices:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 8 & 15 & 9 \\ 4 & 13 & 4 & 13 \\ 8 & 2 & 9 & 15 \end{pmatrix}$$

$$V^{-1} = \begin{pmatrix} 13 & 15 & 16 & 8 \\ 13 & 8 & 1 & 15 \\ 13 & 2 & 16 & 9 \\ 13 & 9 & 1 & 2 \end{pmatrix}$$

$$L_C^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 15 & 1 & 0 & 0 \\ 16 & 7 & 1 & 0 \\ 15 & 13 & 9 & 1 \end{pmatrix}$$

$$U_C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 6 & 13 & 7 \\ 0 & 0 & 6 & 7 \\ 0 & 0 & 0 & 9 \end{pmatrix}$$

$$L_R^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 11 & 3 & 0 & 0 \\ 14 & 4 & 3 & 0 \\ 13 & 9 & 1 & 2 \end{pmatrix}$$

$$U_R = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 5 & 4 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

EXAMPLE 4: Ring R_4

The ring is

$\mathbb{Z}[\omega]$, ω primitive sixteenth root of unity, modulo 17.

The minimum polynomial of ω is

$$f(x) = x^8 + 1.$$

The roots of $f(x)$ modulo 17 are

$$3, 10, 5, 11, 14, 7, 12, 6.$$

Ring R_4 with basis B_{41}

The basis is

$$\{1, \omega, \omega^2, \omega^3, \omega^4, \omega^5, \omega^6, \omega^7\}.$$

Conversion Matrices:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 10 & 5 & 11 & 14 & 7 & 12 & 6 \\ 9 & 15 & 8 & 2 & 9 & 15 & 8 & 2 \\ 10 & 14 & 6 & 5 & 7 & 3 & 11 & 12 \\ 13 & 4 & 13 & 4 & 13 & 4 & 13 & 4 \\ 5 & 6 & 14 & 10 & 12 & 11 & 3 & 7 \\ 15 & 9 & 2 & 8 & 15 & 9 & 2 & 8 \\ 11 & 5 & 10 & 3 & 6 & 12 & 7 & 14 \end{pmatrix}$$

$$V^{-1} = \begin{pmatrix} 15 & 5 & 13 & 10 & 9 & 3 & 1 & 6 \\ 15 & 10 & 1 & 12 & 8 & 11 & 13 & 3 \\ 15 & 3 & 4 & 11 & 9 & 12 & 16 & 10 \\ 15 & 6 & 16 & 3 & 8 & 10 & 4 & 5 \\ 15 & 12 & 13 & 7 & 9 & 14 & 1 & 11 \\ 15 & 7 & 1 & 5 & 8 & 6 & 13 & 14 \\ 15 & 14 & 4 & 6 & 9 & 5 & 16 & 7 \\ 15 & 11 & 16 & 14 & 8 & 7 & 4 & 12 \end{pmatrix}$$

$$L_C^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 14 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 10 & 16 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 12 & 4 & 5 & 1 & 0 & 0 & 0 & 0 \\ 3 & 3 & 7 & 2 & 8 & 1 & 0 & 0 & 0 \\ 13 & 16 & 5 & 10 & 14 & 1 & 1 & 0 & 0 \\ 14 & 8 & 7 & 4 & 12 & 2 & 6 & 1 & 0 \end{pmatrix}$$

$$U_C = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 7 & 2 & 8 & 11 & 4 & 9 & 3 & 3 \\ 0 & 0 & 7 & 8 & 10 & 5 & 1 & 5 & 5 \\ 0 & 0 & 0 & 14 & 5 & 10 & 7 & 5 & 5 \\ 0 & 0 & 0 & 0 & 15 & 11 & 7 & 9 & 9 \\ 0 & 0 & 0 & 0 & 0 & 8 & 3 & 13 & 13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 15 & 4 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 10 \end{pmatrix}$$

$$L_R^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 14 & 3 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 16 & 8 & 6 & 11 & 0 & 0 & 0 & 0 & 0 \\ 8 & 11 & 15 & 6 & 8 & 0 & 0 & 0 & 0 \\ 11 & 11 & 3 & 13 & 1 & 15 & 0 & 0 & 0 \\ 2 & 9 & 6 & 12 & 10 & 8 & 8 & 0 & 0 \\ 15 & 11 & 16 & 14 & 8 & 7 & 4 & 12 & 12 \end{pmatrix}$$

$$U_R = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 10 & 6 & 4 & 3 & 11 & 15 & 15 \\ 0 & 0 & 1 & 6 & 16 & 8 & 5 & 8 & 8 \\ 0 & 0 & 0 & 1 & 4 & 8 & 9 & 4 & 4 \\ 0 & 0 & 0 & 0 & 1 & 3 & 5 & 4 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 11 & 8 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 15 & 15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Ring R_4 with basis B_{42}

The basis is

$$\{1, \sqrt{2+\sqrt{2}}, \sqrt{2}, \sqrt{2-\sqrt{2}}, i, \sqrt{2+\sqrt{2}i}, \sqrt{2}i, \sqrt{2-\sqrt{2}i}\}.$$

Change of Basis Matrices:

To Standard Basis

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

From Standard Basis

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

The fractions in the second matrix indicate that this basis does not generate all of R_4 ; B_{42} is not an integral basis of R_4 .

Conversion Matrices:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 9 & 5 & 12 & 8 & 8 & 12 & 5 & 9 \\ 11 & 6 & 6 & 11 & 11 & 6 & 6 & 11 \\ 5 & 8 & 9 & 12 & 12 & 9 & 8 & 5 \\ 13 & 4 & 13 & 4 & 13 & 4 & 13 & 4 \\ 15 & 3 & 3 & 15 & 2 & 14 & 14 & 2 \\ 7 & 7 & 10 & 10 & 7 & 7 & 10 & 10 \\ 14 & 15 & 15 & 14 & 3 & 2 & 2 & 3 \end{pmatrix}$$

$$V^{-1} = \begin{pmatrix} 15 & 8 & 6 & 12 & 9 & 15 & 7 & 14 \\ 15 & 12 & 11 & 9 & 8 & 3 & 7 & 15 \\ 15 & 5 & 11 & 8 & 9 & 3 & 10 & 15 \\ 15 & 9 & 6 & 5 & 8 & 15 & 10 & 14 \\ 15 & 9 & 6 & 5 & 9 & 2 & 7 & 3 \\ 15 & 5 & 11 & 8 & 8 & 14 & 7 & 2 \\ 15 & 12 & 11 & 9 & 9 & 14 & 10 & 2 \\ 15 & 8 & 6 & 12 & 8 & 2 & 10 & 3 \end{pmatrix}$$

$$L_C^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 12 & 8 & 1 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 14 & 1 & 0 & 0 & 0 \\ 8 & 2 & 1 & 7 & 9 & 1 & 0 & 0 \\ 10 & 2 & 14 & 2 & 13 & 14 & 1 & 0 \\ 5 & 14 & 2 & 4 & 14 & 12 & 9 & 1 \end{pmatrix}$$

$$U_C = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 13 & 3 & 16 & 16 & 3 & 13 & 0 \\ 0 & 0 & 4 & 14 & 14 & 4 & 0 & 0 \\ 0 & 0 & 0 & 12 & 12 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 8 & 9 & 8 \\ 0 & 0 & 0 & 0 & 0 & 15 & 7 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \end{pmatrix}$$

$$L_R^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 15 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 16 & 5 & 13 & 0 & 0 & 0 & 0 & 0 \\ 13 & 1 & 12 & 10 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 11 & 2 & 0 & 0 & 0 \\ 13 & 16 & 8 & 5 & 4 & 8 & 0 & 0 \\ 5 & 1 & 7 & 1 & 15 & 7 & 9 & 0 \\ 15 & 8 & 6 & 12 & 8 & 2 & 10 & 3 \end{pmatrix}$$

$$U_R = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 12 & 13 & 13 & 12 & 1 & 0 \\ 0 & 0 & 1 & 12 & 12 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 16 & 1 & 16 \\ 0 & 0 & 0 & 0 & 0 & 1 & 5 & 13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Observation about the B_{42} basis

The conversion matrix given above for the B_{42} basis factors in a natural way:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 9 & 5 & 12 & 8 & 8 & 12 & 5 & 9 \\ 11 & 6 & 6 & 11 & 11 & 6 & 6 & 11 \\ 5 & 8 & 9 & 12 & 12 & 9 & 8 & 5 \\ 13 & 4 & 13 & 4 & 13 & 4 & 13 & 4 \\ 15 & 3 & 3 & 15 & 2 & 14 & 14 & 2 \\ 7 & 7 & 10 & 10 & 7 & 7 & 10 & 10 \\ 14 & 15 & 15 & 14 & 3 & 2 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 5 & 8 & 9 & 12 & 0 & 0 & 0 & 0 \\ 6 & 11 & 11 & 6 & 0 & 0 & 0 & 0 \\ 8 & 12 & 5 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 5 & 8 & 9 & 12 \\ 0 & 0 & 0 & 0 & 6 & 11 & 11 & 6 \\ 0 & 0 & 0 & 0 & 8 & 12 & 5 & 9 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 13 & 0 \\ 0 & 0 & 0 & 4 & 13 & 0 & 0 & 0 & 0 \\ 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 13 & 0 & 0 & 4 & 0 & 0 & 0 \end{pmatrix}.$$

The 4×4 blocks in the left matrix of the product are the conversion matrices for the B_{12} bases. This is because the B_{42} basis can be written as the union of a B_{12} basis and a multiple of a B_{12} basis:

$$B_{42} = \left\{ 1, \sqrt{2+\sqrt{2}}, \sqrt{2}, \sqrt{2-\sqrt{2}}, i, \sqrt{2+\sqrt{2}i}, \sqrt{2}i, \sqrt{2-\sqrt{2}i} \right\}$$

$$\left\{ 1, \sqrt{2+\sqrt{2}}, \sqrt{2}, \sqrt{2-\sqrt{2}} \right\} \cup \left\{ i, \sqrt{2+\sqrt{2}i}, \sqrt{2}i, \sqrt{2-\sqrt{2}i} \right\}.$$

Because of this basis factorization, two 4×4 conversions could be used instead of one 8×8 conversion. The 4×4 conversions can occur in parallel, but need to be followed by some simple modulo 17 additions representing multiplication by the sparse right matrix of the product. The factorization of bases and the corresponding factorization of the conversion matrices will be the subject of further investigation.

LIST OF REFERENCES

1. A. L. Bequillard and S. D. O'Neil, "Systolic RNS Computation of the 2-Dimensional Discrete Cosine Transform in a Ring of Algebraic Integers," *Proc. of the 20th Annual Conference on Inform. Sciences and Systems*, 783-789, Princeton, NJ, March 1986.
2. D. O. Carhoun, "Inversion of Integral Matrices Using Residue Number System Computation," *Proc. of the 20th Annual Conference on Inform. Sciences and Systems*, 793-798, Princeton, NJ, March 1986.
3. J. H. Cozzens and L. A. Finkelstein, "Computing the Discrete Fourier Transform Using Residue Number Systems in a Ring of Algebraic Integers," *IEEE Trans. Inform. Theory IT-31* (September 1985), 580-588.
4. J. H. Cozzens and L. A. Finkelstein, "Range and Error Analysis for a Fast Fourier Transform Computed over $\mathbb{Z}[\omega]$," *IEEE Trans. Inform. Theory IT-33* (July 1987), 582-590.
5. A. M. Despain, A. M. Peterson, O. S. Rothaus, and E. H. Wold, "Fast Fourier Transform Processors Using Gaussian Residue Arithmetic," *Jour. Parallel and Distributed Processing 2* (1985).
6. R. A. Games, "Complex Approximations Using Algebraic Integers," *IEEE Trans. on Inform. Theory IT-31* (September 1985), 565-579.
7. R. A. Games, "An Algorithm for Complex Approximations in $\mathbb{Z}[e^{2\pi i/8}]$," *IEEE Trans. on Inform. Theory IT-32* (September 1986), 603-607.
8. L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley Publ. Co., Reading, MA, 1985.
9. G. H. Golub and C. F. Van Loan, *Matrix Computations*, John Hopkins Univ. Press, Baltimore, 1983.
10. N. S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.
11. H. T. Kung, "Why Systolic Architectures?," *Computer* (January 1982), 37-46.
12. S. Lang, *Algebra*, Addison-Wesley, Reading, MA, 1972.
13. S. H. Leung, "Application of Residue Number Systems to Complex Digital Filters," *Proc. 15th Asilomar Conf. on Circuits, Systems and Computers* (November 1981), Pacific Grove, CA.
14. M. W. Marcellin and T. R. Fischer, "Encoding Algorithms for Complex Approximations in $\mathbb{Z}[e^{2\pi i/8}]$," *IEEE Trans. on Inform. Theory* (to appear).

15. I. Niven and H. Zuckerman, *An Introduction to the Theory of Numbers*, John Wiley & Sons Inc., New York, 1972.
16. A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," *IEEE Trans. Acoust., Speech, Signal Processing ASSP-27* (December 1974), 456-462.
17. S. K. Rao, "Regular Iterative Algorithms and Their Implementation on Processor Arrays," Ph.D. Thesis, Stanford Univ., October 1985.
18. M. A. Soderstrand, C. Vernia, and J.-H. Chang, "An Improved Residue Number System Digital-to-Analog Converter," *IEEE Trans. Circuits Syst. CAS-30* (December 1983), 903-907.
19. M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, editors, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, New York, 1986.
20. G. W. Stewart, *Introduction to Matrix Computations*, Academic Press, Orlando, 1973.
21. F. J. Taylor and A. S. Ramnarayanan, "An Efficient Residue-to-Decimal Converter," *IEEE Trans. Circuits Syst. CAS-28* (December 1981), 1164-1169.
22. T. Van Vu, "Efficient implementations of the Chinese Remainder Theorem for sign detection and residue decoding," *IEEE Trans. Comput. C-34* (July 1985), 646-651.



MISSION of Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C³I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.